UPC-Universidad Politécnica de Cataluña

Mantenimiento predictivo de los equipos industriales mediante el uso de la inteligencia artificial

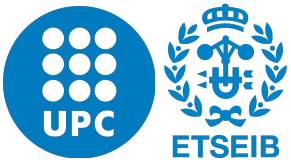
por

Rubén Bartolomé Aramburu

dirigido por:

Cecilio Angulo Bahón

Trabajo de Fin de Grado
Grado en Ingeniería en Tecnologías Industriales



en

ETSEIB
Escuela Técnica Superior de Ingeniería Industrial de Barcelona

Junio 2018

"Artificial Intelligence is the New Electricity."

Andrew Ng

UPC-Universidad Politécnica de Cataluña

Resumen

ETSEIB
Escuela Técnica Superior de Ingeniería Industrial de Barcelona

by Rubén Bartolomé Aramburu

El ser humano siempre ha empleado la inteligencia para tomar decisiones, su condición biológica acota su capacidad de procesamiento y en consecuencia el progreso se demora. Hasta el momento en que se encuentran alternativas. Con el avance estrepitoso de la tecnología, las máquinas tienen cada vez más autonomía, llegando al estado de contar con su propia inteligencia. ¿Para que se dotan a las máquinas de capacidad para aprender por si solas? Por interés. Interés en optimizar los procesos, en crear los modelos más eficientes y en reducir costes. En sintonía con dichos intereses se encuentra la Industria 4.0, aprovechando todos los recursos disponibles para su potenciación.

Se ha aprovechado este contexto para adentrarse en el campo del *Machine Learning*, que tiene un gran potencial. En primer lugar se ha realizado un análisis exploratorio del conjunto de datos disponible. Luego se han realizado los cambio pertintes, en base a la información extraída. A continuación se han creado diversos modelos de clasificación binaria utilizando los algoritmos de regresión logística y *Support Vector Machines* para analizar su comportamiento y extraer el modelo más eficiente.

Por último se ha realizado una comparativa entre los modelos escogidos para cada algoritmo.

Se ha determinado que el sistema obtenido clasifica de forma satisfactoria el conjunto de datos y se ha destacado el hecho, de que según los objetivos, un modelo será mas o menos adecuado que otro.

Agradecimientos

Después de un intenso período de seis meses, hoy es el día: escribo este apartado de agradecimientos para finalizar mi trabajo de fin de grado. Me gustaría nombrar a mi tutor, Cecilio Angulo, para agradecer su supervisión y ayuda cuando la he necesitado. Destacar el papel notable que ha representado mi hermano en este proyecto. Desde la transmisión del potencial que representa el ámbito del Machine Learning hasta el apoyo incondicional en cualquier momento. Muchas gracias por todo.

Glosario

- **Big data:** hace referencia a un conjunto de datos tan grandes que aplicaciones informáticas tradicionales de procesamiento de datos no son suficientes para tratar con ellos y los procedimientos usados para encontrar patrones repetitivos dentro de esos datos.
- **Support Vector Machines:** algoritmo de aprendizaje supervisado que puede ser utilizado tanto para problemas de clasificación como de regresión. Este algoritmo traza cada dato como si se tratase de un punto en un espacio n-dimensional, siendo n el nº de parámetros que se tiene y cada uno de estos el valor de una coordenada.
- **Internet de las cosas (IoT):** es un concepto que se refiere a la interconexión digital de objetos cotidianos con Internet.
- **Internet industrial de las cosas (IIoT):** Los objetos que se relacionan en el IoT son de ámbito industrial
- **Machine Learning:** es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos
- **Industria 4.0:** corresponde a una nueva manera de organizar los medios de producción. El objetivo que pretende alcanzarse es la puesta en marcha de un gran número de «fábricas inteligentes» («smart factories») capaces de una mayor adaptabilidad a las necesidades y a los procesos de producción, así como a una asignación más eficiente de los recursos, abriendo así la vía a una nueva revolución industrial o Cuarta revolución industrial
- **Cloud computing:** conocida también como servicios en la nube, informática en la nube, nube de cómputo, nube de conceptos o simplemente "la nube", es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.
- **Ciberseguridad:** es el área relacionada con la informática y la telemática que se enfoca en la protección de la infraestructura computacional y todo lo relacionado con esta y, especialmente, la información contenida en una computadora o circulante a través de las redes de computadoras
- **Ciberataque:** Intento de vulnerabilizar sistemas informáticos mediante técnicas que aprovechan fallos del sistema o de la seguridad
- **Smart Factory:** Industria o fábrica que aprovecha los sistemas informáticos para automatizar y personalizar los productos generados.

- **Regresión Logística:** tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica (una variable que puede adoptar un número limitado de categorías) en función de las variables independientes o predictoras
- **Deep learning:** es un conjunto de algoritmos de clase aprendizaje automático (en inglés, *machine learning*) que intenta modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no lineales múltiples
- **Inteligencia Artificial (IA):** es la inteligencia exhibida por máquinas. En ciencias de la computación, una máquina «inteligente» ideal es un agente racional flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea
- **Clase (Negativa y positiva):** Conjunto específico dentro de un dataset que generalmente se trata de una variable binaria**.**
- **Función de costes:** representa la suma global de errores cometidos en un conjunto de datos de entrenamiento

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La industria 4.0 está aquí, y con ella arrastra una marea de digitalización, que lleva por estandarte el *Big Data* y una nueva organización de los medios de producción. Y es que en el mundo globalizado, donde conviven las industrias, la competencia es feroz.

Las sociedades están experimentando el frenético avance de las nuevas tecnologías. Ciertas tecnologías que se utilizan a diario se están implementando en las cadenas de producción.

Uno de los campos, que tiene un papel notable en este contexto, se trata del aprendizaje automático. Se estudian las técnicas que permiten a las máquinas aprender por si solas. Dichas máquinas suponen una reducción de costes y una gran fuente de información para la toma de decisiones.

Concretando más hacia este proyecto, el aprendizaje automático puede ser aplicado para la creación de modelos estadísticos que sirvan para el mantenimiento predictivo de un equipo industrial. Dicho mantenimiento se basa en analizar y medir el desgaste de los elementos para sustituirlos en cuanto muestran síntomas que predicen el fallo. De esta manera se evita que se produzca una avería.

Mediante el entrenamiento de algoritmos de aprendizaje supervisado (regresión logística y *Support Vector Machines*) se han generado modelos para la clasificación binaria. Debido a la imposibilidad de contar con un conjunto de datos relativos a un equipo industrial, se ha utilizado un conjunto de datos accesible. Este último presenta un gran número de transacciones electrónicas y algunas de ellas son clasificadas como fraudulentas. El objetivo es entonces crear un modelo estadístico que permita clasificar las transacciones según sean fraudulentas o no. El principio es idéntico al que se debería plantear para el mantenimiento predictivo, pero con otro conjunto de datos.

Con este proyecto se pretende probar distintos modelos a un conjunto de datos, para poder conocer su eficiencia como clasificadores.

Capítulo 2

Descripción del problema y características del proyecto

En este apartado se detallarán los aspectos que se esperan de este proyecto, tanto los objetivos como el alcance. Asimismo, se presentará el contexto en el que se encuentra el proyecto.

2.1. Origen del proyecto

A día de hoy nos encontramos inmersos en la Industria 4.0, lo que brinda una gran cantidad de información en formato de datos masivos o *Big Data* a las empresas de la industria. Estos datos, recopilados por los numerosos sensores que presenta la industria actual, pueden resultar en informaciones valiosas para optimizar numerosos procesos industriales. Por lo tanto el *Big Data* explotado de forma adecuada, puede resultar en un ahorro considerable de recursos.

Para el ser humano puede tratarse de una tarea muy complicada el hecho de tener que analizar tal información masiva, por ello acudimos a la computación. En este caso, se diseñan modelos predictivos, que mediante los datos pueden anticipar situaciones que sean indeseables a la hora de producir.

El proyecto que describe este informe tratará sobre el mantenimiento predictivo de un proceso industrial, a partir de los datos que se cuenten de los equipos industriales que se vean involucrados, el modelo deberá predecir si la máquina fallará y en consecuencia sea necesario un reajuste de la maquinaria.

Existen diferentes modelos para la clasificación de clases, es importante escoger algoritmos adecuados para el conjunto de datos. Este proyecto trata un problema de clasificación binario, se utilizarán las técnicas de regresión logística y SVM para sacar conclusiones de los datos que se deban analizar.

2.2. Motivación

La motivación de este proyecto es la de ajustar varios modelos de clasificación, para ver cual es el más adecuado para el mantenimiento predictivo.

Son muchas las variables que forman parte del correcto funcionamiento de un equipo industrial. Según el estado de dichas variables, el equipo puede llegar a fallar. Algunas posibles causas serían: ajustes inadecuados, engrase incorrecto, desequilibrio de los pares de apriete...

Un operario, monitorizando dichas variables, es capaz de determinar anomalías. Por otra parte, a diferencia de los algoritmos de clasificación, no es capaz de resaltar patrones entre las diversas variables que puedan provocar un fallo. Se deben minimizar los paros de emergencia y tiempos muertos, causando impacto financiero negativo.

Por otro lado, existe también una motivación académica. Sería gratificante realizar un trabajo que refleje que, los conocimientos que se han ido adquiriendo a lo largo de estos años de etapa universitaria, pueden ser aplicados en el mundo laboral para mejorar ciertos aspectos del mismo.

Finalmente, existe una motivación personal por enriquecer el conocimiento en el ámbito del Machine Learning, debido a su gran potencial.

2.3. Contexto

El marco de este proyecto corresponde al de la Industria 4.0. La cuarta revolución industrial reside en una nueva manera de organizar los medios de producción, adaptados a las nuevas tecnologías. La Industria 4.0 es un concepto generado y promovido por el gobierno y por la industria alemana para fomentar, mantener y aumentar la capacidad industrial y productiva del país. Está alineada con las políticas de otros países occidentales (*e.g.* programa AMP de USA). En resumidas cuentas, la Industria 4.0 consiste en cambiar los sistemas y métodos de fabricación utilizando nuevas tecnologías para conseguir fábricas inteligentes (*Smart Factories*) más productivas, más flexibles y más eficientes.

Teniendo en cuenta la alta competitividad industrial y las nuevas tecnologías presentes en la sociedad aplicables a la industria, adaptarse al cambio parece irreversible. Y es que para poder competir en un un mundo globalizado, con países donde la mano de obra es sumamente económica debido a los derechos laborables prácticamente ausentes, se deben utilizar todos los recursos disponibles.

¿A que se refiere el término 4.0?

A día de hoy el marco europeo se sitúa en la cuarta revolución industrial. Para comprender los avances que presenta la revolución actual, a continuación se repasarán las últimas tres revoluciones industriales.

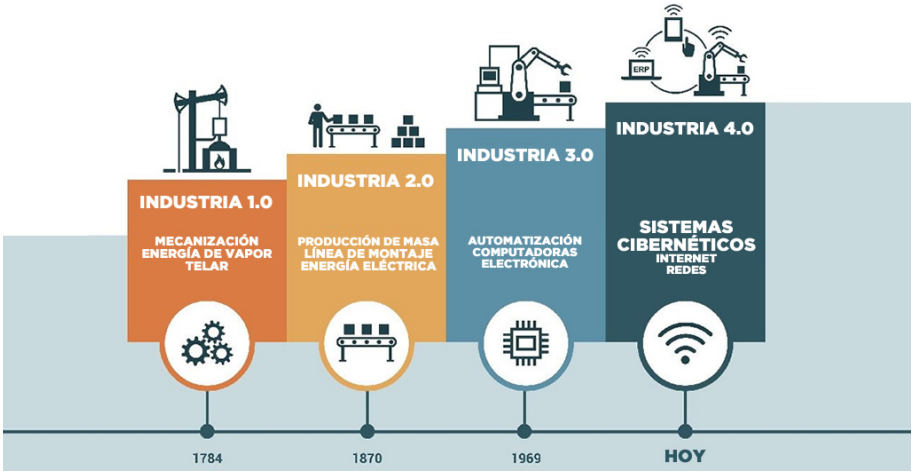


FIGURA 2.1: Evolución de la industria

A finales del siglo 18, la primera revolución industrial llegó a Gran Bretaña con la introducción de las máquinas para la producción. Se dio el salto de la producción manual al uso de motores de vapor y agua como fuentes de energía.

Dichos avances fueron muy útiles para la agricultura y el término "fábrica" comenzó a escucharse con mayor frecuencia. En aquella época la industria textil formaba gran parte de la economía británica y fue dicha industria la primera en adoptar los métodos de la revolución industrial.

La segunda revolución industrial introdujo sistemas que ya existían en las industrias, como serían los telégrafos y ferrocarriles. La característica fundamental de dicha revolución es la introducción de la producción en masa como metodología a seguir en los entornos productivos.

La implantación de la electricidad en las fábricas contribuyó al incremento de las tasas de producción. La producción masiva de acero ayudó a introducir los ferrocarriles en el sistema, lo que consecuentemente contribuyó a la producción en masa. Por otra parte,

también se dio un gran salto en el sector de la química con invenciones como el tinte sintético.

La tercera revolución industrial suele hacer referencia a la revolución digital. Se cambiaron los sistemas analógicos y mecánicos por sistemas digitales. Esta revolución es el resultado del gran desarrollo en computadoras, tecnología de información y comunicación.

La cuarta revolución industrial lleva la automatización de los procesos de fabricación a un nivel superior, introduciendo tecnologías de producción en masa personalizadas y flexibles.

Esto significa que las máquinas operarán de manera independiente o cooperarán con los humanos con la premisa de crear un campo de producción orientado al cliente. Dicho entorno se preocupa por auto-mantenerse. La máquina se convierte en una entidad independiente que puede recopilar datos, analizarlos y asesorar sobre ellos.

La concepción de dicho entorno requiere implantar las tecnologías de auto-optimización, auto-cognición y auto-personalización en la industria. De esta manera los fabricantes podrán comunicarse con las computadoras en lugar de operarlas.

¿Cuales son los pilares de la Industria 4.0?

FIGURA 2.2: Los pilares de la Industria 4.0

La nueva Industria 4.0 tiene varios ejes entorno a los que se articula y en los que el fabricante tendrá que trabajar para integrar en sus plantas de producción.

A continuación se detallan 9 pilares fundamentales de la Industria 4.0:

- **El Internet Industrial de las Cosas.**

  El IIoT o Internet Industrial de las Cosas, se trata del uso del IoT en la manufactura. Donde IoT o Internet de las Cosas se refiere a la interconexión digital de objetos a Internet. Se incorpora *Machine Learning* en los equipos industriales y se explota el *Big Data* aprovechando los datos de los sensores. Otro concepto importante en el marco del IIoT es la comunicación máquina-a-máquina (M2M) que agiliza los intercambios de datos.

  El principio es que las máquinas inteligentes son mejores que los seres humanos en la captura y comunicación de datos con precisión y coherencia. Con una correcta explotación de los datos, se pueden captar de antemano las ineficiencias y problemas, para actuar en consecuencia.

  El IIoT tiene un gran potencial en lo que respecta la fabricación. Desde el control de calidad hasta las prácticas sostenibles, aplicar el IoT en la manufactura promueve una mayor eficiencia de la cadena de suministro.

  Uno de los desafíos más relevantes en el IIoT es la interoperabilidad. Este es el problema que surge debido a que las diferentes máquinas y dispositivos utilizan protocolos y arquitecturas diferentes. Por ello, la tendencia está en la creación de estándares y el desarrollo de arquitecturas comunes.

- **Simulación.**

  Las simulaciones se usarán más extensivamente en las operaciones de planta. Se aprovecharán los datos en tiempo real para reflejar el mundo físico en un modelo virtual, que puede incluir máquinas, productos y humanos. Los operadores podrán probar y optimizar la configuración de la máquina, para el próximo producto que será procesado, en el mundo virtual antes del cambio físico. El uso de la simulación reduce los tiempos de configuración de las máquinas y aumenta la calidad en la ejecución de las operaciones.

  Simular los modelos virtuales, previamente al cambio físico es fundamental para controlar los costes futuros además de garantizar la calidad y la eficiencia en el desarrollo de productos.

- ***Big Data.***

  Big Data es la gestión y análisis de enormes volúmenes de datos que no pueden ser tratados de manera convencional. Debido a la ingente cantidad de datos que se reciben de los números sensores integrados en los procesos, se debe tratar dicha información. Estos superan las capacidades de las herramientas de software habitualmente utilizadas para la captura, gestión y procesamiento de datos.

Convertir los datos masivos en información que facilita la toma de decisiones es el objetivo del *Big Data*.

La recolección y análisis de los datos provenientes de diferentes equipos y sistemas de producción así como de los sistemas de gestión empresarial y de clientes, se convertirán en esenciales para la toma de decisiones en tiempo real.

- **Robots autónomos.**

  Los robots autónomos son capaces de trabajar para automatizar y coordinar una serie de tareas logísticas y de producción, sin la necesidad de ser supervisados por un humano. Dichos robots son capaces de interactuar entre ellos e incluso con humanos, trabajando y aprendiendo de ellos. Puesto que dichos robots son capaces de interactuar con el entorno, se caracterizan de avanzados frente a a los robots que a día de hoy constituyen la mayor parte del entramado industrial.

  El uso de dichos robots reducirá los costes y aumentará la producción, ya que frente a los robots empleados hoy en día, estos contarán con un rango de capacidades superior.

- **Integración vertical y horizontal.**

  La integración horizontal hace referencia a digitalizar la organización interna de la empresa (producción del producto). La integración vertical reside en digitalizar los datos de la actividad de un tercero (por ejemplo: transporte del producto).

  La industria 4.0 propone una mayor armonía entre todos los que forman parte del ecosistema, su objetivo reside en contar con una única plataforma de gestión a la que todos tengan acceso.

- **La nube.**

  El modelo tecnológico del *Cloud Computing* o computación en la nube, sostiene el desarrollo de la industria de 4.0. Para soportar el uso de aplicaciones y datos compartidos entre diferentes ubicaciones y sistemas debido a las numerosas tareas relacionadas con la producción de bienes y servicios, dicha información debe estar alojada de manera que sea accesible. El *Cloud Computing* responde a los requerimientos anteriores y ofrece una gran reducción en el coste, el tiempo y la eficiencia.

- **Ciberseguridad.**

  En el marco de la Industria 4.0, se manejan continuamente cantidades masivas de datos que pueden ser vulnerables a cualquier intromisión. Es por ello que siempre se deberá prestar mucha atención a la ciberseguridad.

La ciberseguridad hace referencia a la seguridad de la información electrónica. Este área se dedica a defender a las computadoras y los servidores, los sistemas electrónicos, las redes y los datos frente a posibles ataques maliciosos. Por otra parte, además de mirar por la seguridad informática, se tratan situaciones de recuperación ante desastres y educación del usuario final.

No sólo se debe potenciar la seguridad informática sino que también se debe prestar suma atención a la seguridad de la información. Todo esto con el objetivo de proteger toda la estructura telemática de las vulnerabilidades.

La tendencia que viven las fábricas en transformarse en *Smart Factories* implica que gran parte de los sistemas de control de las fábricas deben ser acondicionados para la nueva tecnología. Se debe prestar mucha atención a dicha adaptación puesto que la mayoría de equipos aún corren en sistemas operativos antiguos y hasta ahora muchos no solían estar conectados a Internet. Esta transformación implica un elevado riesgo a ciberataques.

La ciberseguridad se dedica a prevenir estas acciones intrusivas. Las redes son protegidas mediante cortafuegos y software de protección de acceso a servidores y nubes.

- **Realidad aumentada.**

  La realidad aumentada hace referencia a la tecnología que permite superponer información digital sobre una imagen real del entorno en el que se encuentra el usuario.

  A pesar de ser uno de los pilares menos desarrollados de la industria 4.0, la realidad aumentada es compatible con una gran variedad de aplicaciones y servicios en diferentes campos.

  En un futuro, las empresas utilizarán de manera más frecuente dicha tecnología para la mejora de los procedimientos de trabajo. Los trabajadores contarán con información en tiempo real que propiciará una mejor toma de decisiones y metodología de trabajo.

- **Fabricación aditiva.**

  La fabricación aditiva es la producción de partes de capas de material superpuestas, típicamente en forma de polvo, para obtener un modelo 3D.

  Las empresas comienzan a adoptar la fabricación aditiva (*e.g.* mediante la impresión 3D) para el uso principalmente en la creación de prototipos y para la producción de componentes individuales.

En el marco de la Industria 4.0, los métodos de fabricación aditiva serán frecuentemente utilizados para la producción de pequeños lotes de productos personalizados. La arquitectura de dichos productos puede ser más compleja y ligera ya que se beneficia de las ventajas de dichos métodos.

A través de los ejes presentados anteriormente se puede lograr una mayor flexibilidad en la producción, una gestión correcta de los diferentes elementos de la producción, una integración completa con las diferentes partes y departamentos de la empresa, una reducción de los stocks y plazos de entrega, mejora del *time to market* (TTM) [1], una implementación de mantenimientos predictivos y un mayor ahorro de energía.

En este proyecto, se abordará el mantenimiento predictivo de los equipos industriales, una de las consecuencias de la Industria 4.0. En el siguiente apartado se presentarán sus características y ventajas.

2.3.1. Mantenimiento predictivo

En el marco industrial, el mantenimiento de equipos ha evolucionado de forma considerable debido en gran parte al desarrollo tecnológico de los equipos de control y medida.

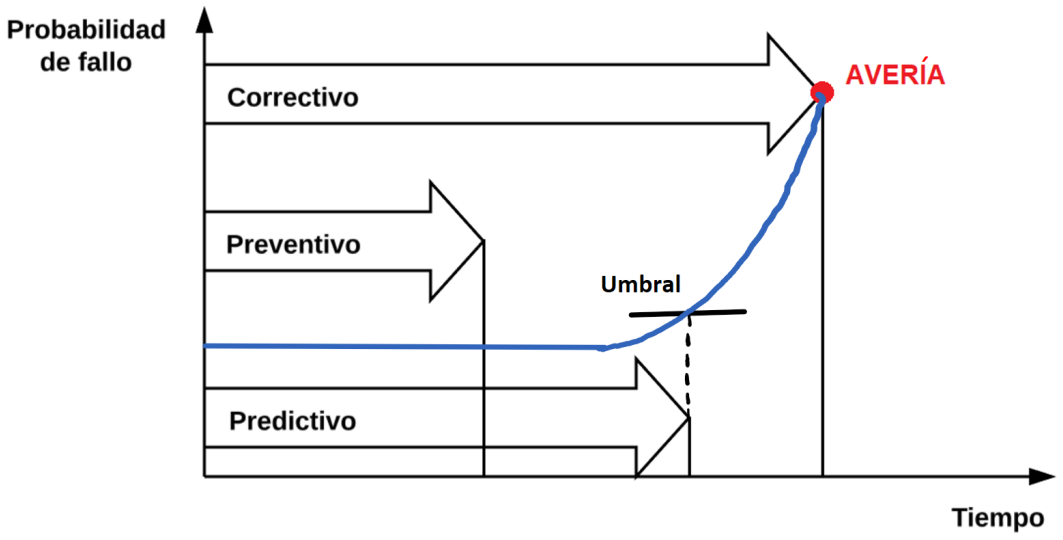


FIGURA 2.3: Tipos de mantenimiento

En la figura 2.3 se aprecian los diferentes tiempos de actuación. El mantenimiento correctivo se aplica en el momento que ocurre una avería, lo que implica tiempos muertos.

[1] Tiempo transcurrido desde que un producto es concebido hasta su introducción en el mercado

La diferencia entre mantenimiento predictivo y preventivo reside en que el mantenimiento predictivo actúa en la situación que el sistema tiende a la avería, determinando el instante adecuado para la intervención. Por lo tanto el mantenimiento predictivo esta más ajustado al comportamiento del equipo.

El mantenimiento correctivo es reparar cualquier equipo en el momento que se detecta algún fallo. Dicho fallo puede poner en riesgo el funcionamiento seguro del equipo y de la persona que lo utiliza.

El mantenimiento preventivo resulta en hacer los ajustes necesarios para mantener cualquier herramienta o equipo en condiciones seguras de uso, con el fin de evitar posibles fallos que impliquen tiempos muertos de corrección.

El mantenimiento predictivo, aplicado a los equipos industriales, evalúa el estado de la maquinaria basándose en su condición, de manera que recomienda intervenir o no. Mediante sensores, se extraen datos que luego son procesados para conocer la condición del equipo. A través de un modelo de clasificación, se determina si es necesaria la intervención, dicho modelo capta los modos de fallos potenciales de los equipos. Mediante dicho mantenimiento se logra optimizar la fiabilidad y disponibilidad de la maquinaria al mínimo coste.

Para crear dicho modelo se necesitan medir parámetros que muestren una relación con el ciclo de vida del equipo. (*e.g.* vibración de los cojinetes, temperatura de las conexiones eléctricas, tiempo de proceso, etc.).

En este proyecto se pretende contar con un conjunto de datos referentes a un equipo industrial, con el objetivo de crear diversos modelos para el posterior mantenimiento predictivo.

2.4. Objetivos del proyecto

Durante el proyecto se aprenderán los pasos necesarios para crear un modelo predictivo, así como su posterior aplicación al *Big Data* detrás de la Industria 4.0. Serán empleados los modelos de Regresión Logística y de SVM para abordar la problemática.

Debido a que no se dispone de un *dataset* correspondiente a un equipo industrial, se ha optado por utilizar un *dataset* con propiedades parecidas al deseado.

Los objetivos del proyecto son los siguientes:

- **Descubrir el campo del *Machine Learning*** Se busca aprender parte de la teoría que conforma los algoritmos de aprendizaje supervisado. El análisis de datos

para su posterior procesamiento. El uso de las herramientas de programación para el analisis.

- textbfImplementación de modelos mediante la librería scikit learn Implementar diversos modelos de regresión logística y support vector machines para analizar su eficiencia. Por otra parte, ajustar hiperparámetros con el objetivo de optimizar los modelos.

- textbfAplicar los conocimientos aprendidos Una vez se haya aprendido a implementar modelos y a ajustarlos, confeccionar un modelo que se ajuste de forma correcta al *dataset* de estudio.

- tstbfContrastar los resultados de los modelos Con los modelos de elección, ser capaz de contrastar los valores de las diferentes métricas para realizar un análisis global de los resultados.

2.5. Alcance del proyecto

Este proyecto pretende crear un modelo de clasificación binario adecuado para el mantenimiento predictivo de un equipo industrial, del cual se cuente con un conjunto de datos para la creación del modelo. Por otra parte, mediante ciertos ajustes, este modelo puede ser utilizado para otros equipos industriales. El objetivo es encontrar el modelo que mejor se ajuste para el mantenimiento predictivo.

Pese a que el objetivo principal sea el antes mencionado, es posible que los modelos que se diseñen puedan ser adaptados para cubrir otras necesidades de los procesos industriales. Una de las posibles aplicaciones sería la identificación de patrones de no calidad. Según la entrega que ofrece un determinado equipo industrial, se puede deducir si las variables del equipo son las adecuadas o se debe actuar sobre ellas para garantizar la calidad.

De hecho, los modelos que se detallarán en este proyecto, pueden ser extrapolados a muchos otros ámbitos. Los modelos predictivos no son exclusivos del sector de la industria, sino que se emplean en la medicina (*e.g.* realizar prediagnósticos médicos), automoción (*e.g.* vehículos autónomos),etc. Por lo tanto los modelos empleados en este proyecto, pueden ser muy útiles en una multitud de campos.

Capítulo 3

Modelos estadísticos de machine learning

Hoy en día se escuchan con mayor frecuencia los términos: Inteligencia Artificial, Machine Learning y Deep Learning, si bien existe una relación entre ellos, cada uno tiene un significado distinto. Para abordar la problemática que se ha planteado anteriormente es necesaria la comprensión de dichos conceptos para su posterior aplicación.

3.1. Introducción

Can machines think? (A.M. Turing, 1950). Fue entonces, en 1950, cuando por primera vez se planteó en un artículo científico la posibilidad de que las máquinas pensaran. En aquella época los recursos con los que se contaba no eran los adecuados para proliferar el concepto, esencialmente por falta de potencia computacional (los ordenadores no contaban con capacidad de almacenar información) y de fondos.

Es importante destacar el contexto en el que se encontraba Turing cuando escribió dicho artículo. En la primera mitad del siglo XX, la ciencia ficción familiarizó al mundo con el concepto de la inteligencia artificial. Y es que los humanoides, que se hacían conocer en la gran pantalla, eran una clara representación del concepto.

La inteligencia artificial consiste en máquinas que pueden realizar tareas que son características de la inteligencia humana (*i.e.* pensar, comprender el lenguaje, reconocer imágenes, planificar, etc). Además, esta puede clasificarse en dos categorías, general y limitada (o *narrow*).

Por un lado, la inteligencia artificial general tiene todas las características de la inteligencia humana, incluyendo las capacidades que se han mencionado anteriormente. Por otro lado, la inteligencia artificial limitada exhibe únicamente ciertas facetas de la inteligencia humana, y puede realizar dichas facetas de forma muy precisa. Una máquina que reconoce adecuadamente las imágenes que se le enseñan sería un ejemplo de inteligencia artificial limitada.

¿Entonces, mediante qué proceso una máquina consigue adoptar dichas capacidades? Mediante el denominado *Machine Learning*, el cual no es más que una manera de conseguir la inteligencia artificial [1]. Una de las grandes ventajas de este método es que evita que deban ser programadas todas las líneas de código que representarían las capacidades, lo que supondría escribir millones de ellas. El planteamiento se basa en entrenar a un algoritmo, para que él mismo sea capaz de aprender de forma autónoma. Este entrenamiento se realiza a base de ingentes cantidades de información que utiliza el algoritmo para realizar sus propios ajustes. En este proyecto se dispondrá de toda la información, que nos aporta el IoT, acerca del desgaste de los elementos que conforman la maquinaria que necesitamos controlar así como de otros datos del entorno.

Las máquinas pueden recibir distintos entrenamientos para atribuirse una inteligencia artificial, el *deep learning* sería uno de ellos. Otros entrenamientos podrían ser el aprendizaje del árbol de decisiones, la programación de lógica inductiva, la agrupación, el aprendizaje de refuerzo o las redes *bayesianas*.

El *deep learning* se inspira de la estructura y función del cerebro humano. Básicamente es una simulación de una red de neuronas. Las redes neuronales artificiales son los algoritmos que simulan la estructura biológica del cerebro humano. El termino "*deep*" (profundo) hace referencia a las diversas capas que atraviesan los *inputs* para proporcionar un *output*.

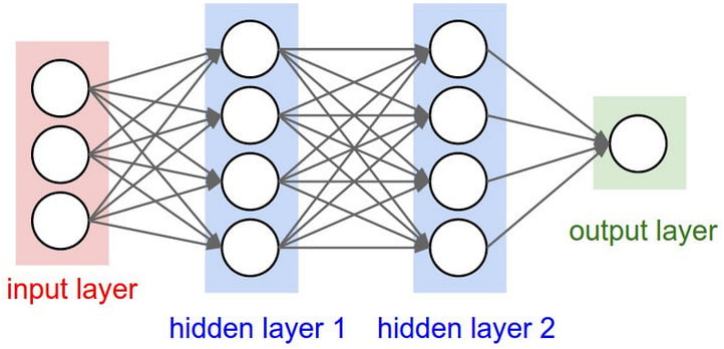


FIGURA 3.1: Estructura red neuronal

[1] *The ability to learn without being explicitly programmed* (A. Samuel, 1959) definiría el concepto.

Como se puede ver en la figura, al igual que en el cerebro humano, las redes neuronales se caracterizan por un numero de capas intermedias entre la entrada de la información y el resultado final.

En este proyecto se tratarán dos algoritmos basados en entrenamientos diferentes: la regresión logística y *Support Vector Machines*. Estos entrenamientos serán descritos más adelante y serán aplicados a un conjunto de datos.

3.2. Tipos de algoritmos

En el marco del machine learning, existen esencialmente tres grandes grupos de algoritmos: los de aprendizaje supervisado (SL) , aprendizaje no supervisado (UL) y los de aprendizaje por refuerzo (RL). Los algoritmos supervisados y no supervisados son los más frecuentes, por otra parte el aprendizaje por refuerzo es muy eficaz a la par que complejo para ser aplicado.

Los algoritmos supervisados intentan modelizar las relaciones y dependencias entre la predicción objetivo y los datos que conocemos, de manera que gracias a las relaciones que se establezcan sea posible realizar predicciones con datos futuros. Dados unos conjuntos de variables de entrada x1, x2, x3 y sus respectivas salidas y, ser capaces de deducir la función f(x1,x2,x3) que modela el comportamiento del sistema. En estos algoritmos conocemos a qué corresponden las variables de entrada y en qué resultan. Se abordarán en este proyecto dos algoritmos supervisados puesto que se cuenta con un conjunto de datos que lo permite.

Por otra parte, los algoritmos no supervisados se difieren de los supervisados debido al desconocimiento de las variables de entrada. Estos algoritmos son útiles cuando no se es capaz de saber qué buscar en los datos. Se utilizan en esencia para buscar patrones en los datos y hallar modelos descriptivos, estos ayudan a obtener ideas significativas y describir mejor los datos a los usuarios.

Por último, el aprendizaje por refuerzo se basa en un agente que trata de aprender un comportamiento mediante interacciones de prueba y error en un ambiente dinámico e incierto. En general, al sistema no se le dice que acción debe tomar, sino que él debe de descubrir qué acciones dan el máximo beneficio. En un RL estándar, un agente esta conectado a un ambiente por medio de percepción y acción. El comportamiento del agente debe de ser tal que seleccione acciones que tiendan a incrementar a largo plazo la suma de las recompensas totales.

Se utilizarán algoritmos basados en el aprendizaje supervisado con el objetivo de crear un modelo predictivo que actúe como un clasificador binario. Dicho modelo será discriminatorio, existirán dos estados: 0 si la máquina no necesita ser asistida (en vistas a evitar un probable fallo futuro) y 1 si la máquina debe ser asistida (reparación o sustitución de alguna pieza).

3.3. Regresión logística

La regresión logística es un algoritmo de aprendizaje supervisado, se utiliza para tareas de clasificación. Se trata de un método estadístico para analizar un conjunto de datos en el que existe una o más variables que determinan una respuesta. La respuesta se mide con una variable dicotómica, esta se utiliza para predecir una respuesta de carácter binario, dado un conjunto de variables independientes. La regresión logística predice la probabilidad de ocurrencia de un evento, mediante la introducción de datos en una función lógica.

En los problemas de clasificación, la variable que se pretende predecir es una variable y que puede tomar dos valores: $y \in \{0, 1\}$. Donde 0 representa la clase negativa y 1 la clase positiva. Para el contexto de este proyecto, la clase positiva representa un reajuste de la máquina necesario y la negativa un reajuste innecesario.

Por otra parte dicho algoritmo puede ser aplicado para clasificación multiclase.

3.3.1. Representación de la hipótesis

El clasificador contará con una hipótesis $h_\theta(x)$ que retornará valores entre 0 y 1. La regresión logística se basa en la regresión lineal. La regresión lineal utiliza la siguiente hipótesis:

$$h_\theta(x) = \theta^T x$$

Donde los parámetros θ se ajustan a los datos de tal forma que se minimice el error global y x corresponde a las variables del modelo. En la siguiente figura se puede observar que la hipótesis viene dada por la recta roja, está puede ser descrita mediante la ecuación anterior.

A diferencia de la regresión logística, la hipótesis de la regresión lineal puede retornar valores inferiores a 0 y superiores a 1. Esto se debe a que para unos valores de x dados, $\theta^T x \in \mathbb{R}$. Para acotar los resultados de la hipótesis de la regresión logística entre 0 y 1,

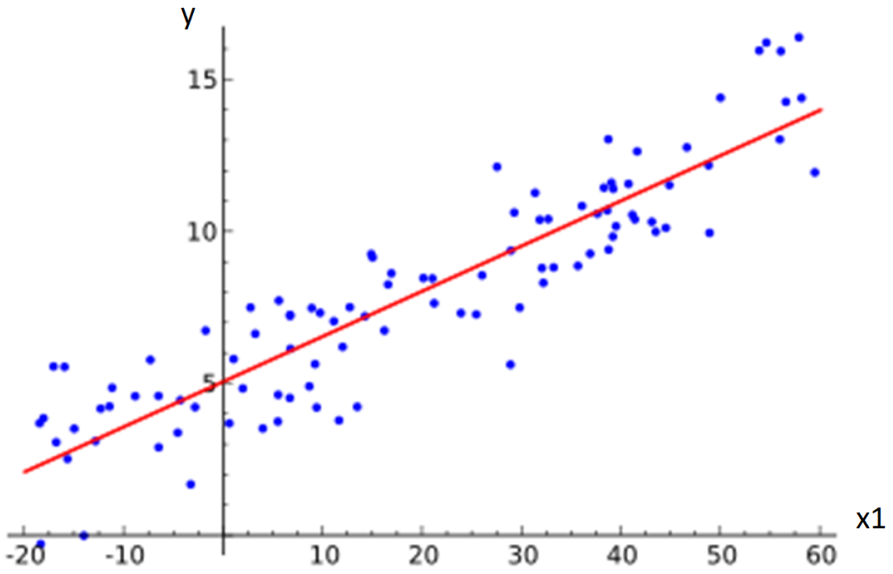


FIGURA 3.2: Ejemplo de regresión lineal

se le aplicará una función a la hipótesis de la regresión lineal. Está función se conoce como la función *sigmoide* o logística y se describe según la siguiente ecuación:

$$g(z) = \frac{1}{(1+e^{-z})}$$

La función *sigmoide* es una función en forma de S que puede tomar cualquier valor real y retorna un valor dentro del rango *(0,1)*. Cómo se puede observar, el intervalo excluye los valores 0 y 1 puesto que la función presenta dos asíntotas horizontales en dichos valores. En la siguiente figura, de la función *sigmoide*, se pueden apreciar dichas características:

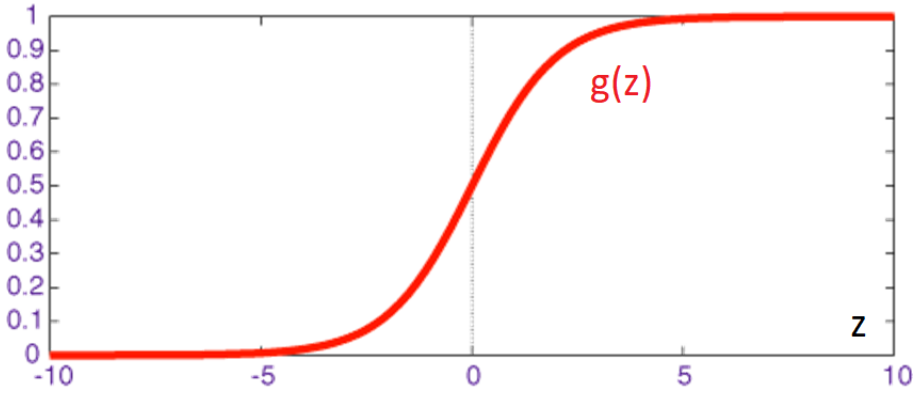


FIGURA 3.3: Función *sigmoide*

Mediante la función *sigmoide* y la hipótesis de la regresión lineal se define la hipótesis de la regresión logística. La regresión logística sigue la siguiente hipótesis:

$$h_\theta(x) = g(\theta^T x)$$

El valor que devuelve la hipótesis, dados los valores de x, corresponde a la probabilidad estimada de que dicha muestra corresponda a la clase positiva 1. Por lo tanto la hipótesis de la regresión logística es equivalente a $P(y = 1|x, \theta)$. (*e.g.* Dados unos valores para x, si el resultado es $h_\theta(x) = 0,7$, esto implicaría que existe un 70 % de probabilidades de que dicha muestra sea de la clase positiva (y=*1*)).

En el siguiente apartado se presentará la metodología empleada para decidir de qué clase es una muestra, mediante el uso del valor que nos retorna la hipótesis de la regresión logística.

3.3.2. Límite de decisión

Se ha conocido en el apartado anterior la hipótesis del modelo de regresión logística. Cabe destacar que dicha hipótesis no retorna un valor binario $y \in \{0, 1\}$ sino que retorna un valor real $y \in (0, 1)$. Entonces, ¿Cómo se realiza la clasificación a partir de los valores que retorna la hipótesis del modelo?

El modelo clasificará como clases positivas ($y = 1$) las muestras que resulten en: $h_\theta(x) \geqslant 0,5$. En su defecto, el modelo clasificará como clases negativas ($y = 0$) las muestras que resulten en $h_\theta(x) < 0,5$.

Por lo tanto, si se extrapola esta clasificación a la definición de la hipótesis, se obtiene lo siguiente:

$$If\ y = 1: \quad h_\theta(x) \geqslant 0,5 \ \Rightarrow\ g(\theta^T x) \geqslant 0,5$$
$$If\ y = 0: \quad h_\theta(x) < 0,5 \ \Rightarrow\ g(\theta^T x) < 0,5$$

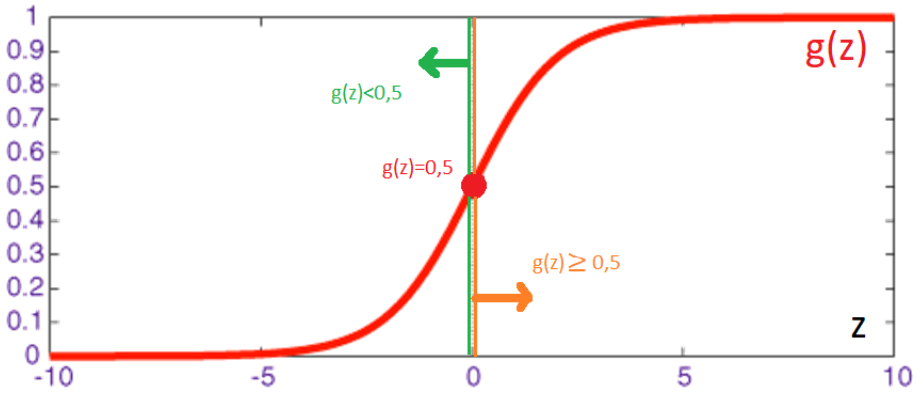


FIGURA 3.4: Propiedades *sigmoide*

Entonces,

$$If\ y = 1: \quad \theta^T x \geqslant 0$$

$$If\ y = 0: \quad \theta^T x < 0$$

A través de las propiedades que presenta la hipótesis del modelo, debido a la influencia de la función *sigmoide*, a continuación se realizarán dos ejemplos. Estos ejemplos facilitarán una mejor intuición acerca de la metodología que emplea el clasificador.

3.3.2.1. Ejemplo límite de decisión lineal

En el siguiente conjunto de datos, las muestras son caracterizadas por las variables x_1 y x_2. Estas pueden ser de la clase negativa (círculo azul) o de la clase positiva (cruz roja). Se aprecia que dichas clases pueden ser separadas de forma lineal.

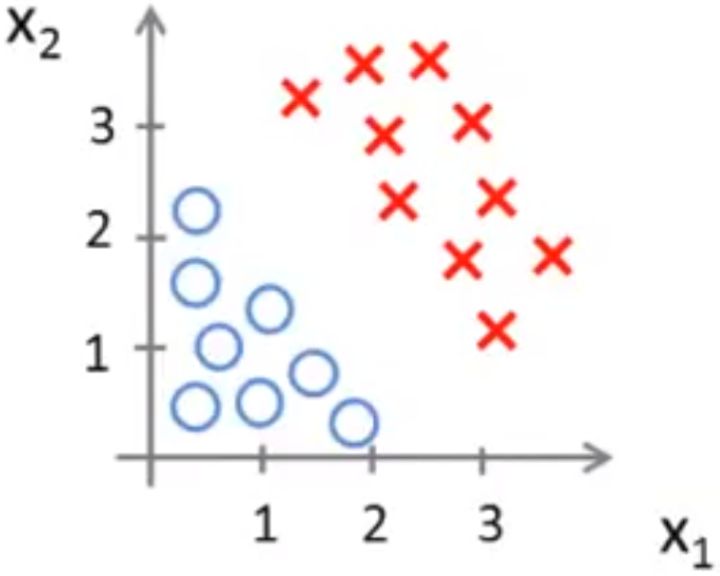


FIGURA 3.5: Conjunto de datos linealmente separable

Se debe hallar la recta que separa ambas clases, para ello se deben encontrar los coeficientes θ que acompañan a cada variable. Más adelante se mostrará la técnica empleada para encontrar dichos coeficientes. Para el siguiente ejemplo, se asume que los coeficientes han sido determinados y son los siguientes:

$$\theta = [-3, 1, 1]$$
$$con, \quad x = [x_0, x_1, x_2] \quad donde\ x_0 = 1$$

entonces,

$$\theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$$

sustituyendo,

$$\theta^T x = -3 + x_1 + x_2$$

El límite de decisión se encuentra entonces para $\theta^T x = 0$ o lo que es equivalente a $-3 + x_1 + x_2 = 0 \Rightarrow x_1 + x_2 = 3$. Representando dicha ecuación en el gráfico anterior, se obtiene la figura 4.6.

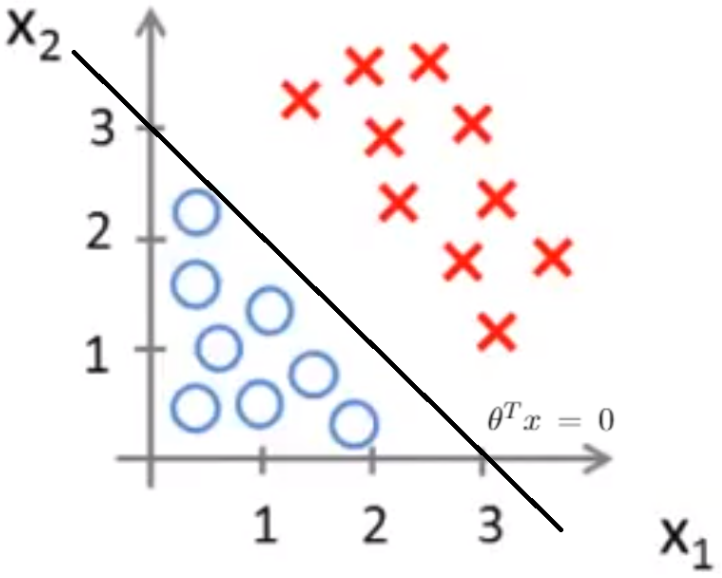


FIGURA 3.6: Límite de decisión

Puesto que se ha definido que para $\theta^T x \geqslant 0 \quad y = 1$, las muestras que caigan en dicha recta serán clasificadas cómo clase positiva. Por lo tanto:

$$\begin{aligned} Clasificar\ y = 1: &\quad \theta^T x \geqslant 0 \Leftrightarrow x_1 + x_2 \geqslant 3 \\ Clasificar\ y = 0: &\quad \theta^T x < 0 \Leftrightarrow x_1 + x_2 < 3 \end{aligned}$$

Según las ecuaciones anteriores, se separan dos zonas, la zona de clasificación positiva (zona roja en la figura 4.7) y la zona de clasificación negativa (zona azul en la figura 4.7).

En resumidas cuentas, la regresión lineal clasifica según la ecuación $\theta^T x$ sea mayor, menor o igual a 0.

3.3.2.2. Ejemplo límite de decisión no-lineal

En el siguiente conjunto de datos, las muestras son caracterizadas por las variables x_1 y x_2. Estas pueden ser de la clase negativa (círculo azul) o de la clase positiva (cruz roja).

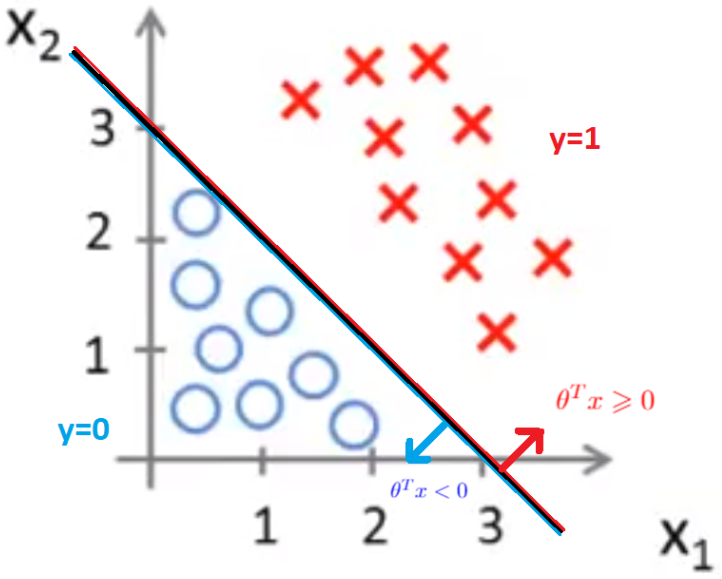


FIGURA 3.7: Zonas de clasificación

En este ejemplo se aprecia que dichas clase no pueden ser separadas de forma lineal. En este apartado se abordarán los límites de decisión no-lineales.

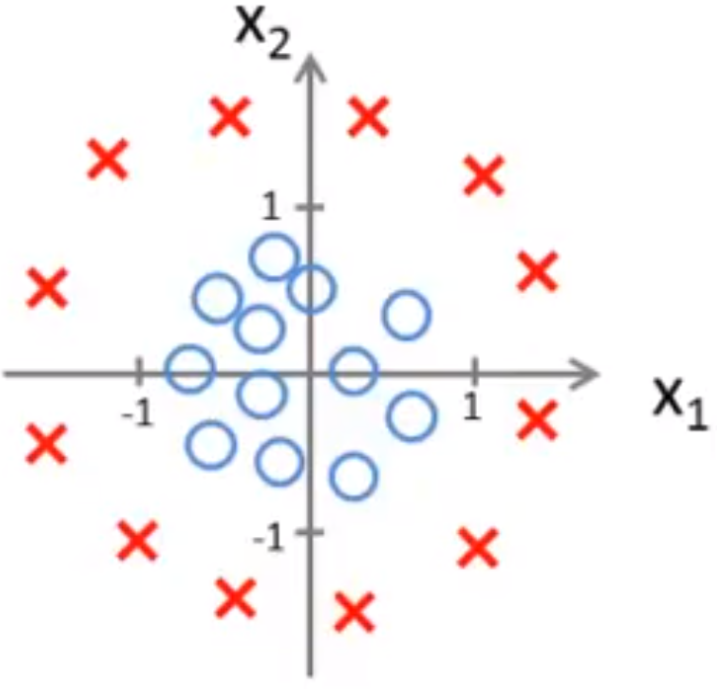


FIGURA 3.8: Conjunto de datos linealmente no separable

Para poder ajustar el límite de decisión al conjunto de datos, es necesario añadir términos de mayor grado a la hipótesis. Los parámetros no serán simplemente las variables que intervienen en su grado menor, sino que serán combinaciones entre variables o grados

mayores de los mismos parámetros. Un ejemplo de hipótesis para resolver el conjunto presentado en la figura 4.8 sería el siguiente:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Análogamente al ejemplo anterior, se asume que los coeficientes han sido determinados y son los siguientes:

$$\theta = [-1, 0, 0, 1, 1]$$
$$con, \quad x = \left[x_0, x_1, x_2, x_1^2, x_2^2\right] \quad donde\ x_0 = 1$$

entonces,

$$\theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$$

sustituyendo,

$$\theta^T x = -1 + x_1^2 + x_2^2$$

El límite de decisión se encuentra entonces para $\theta^T x = 0$ o lo que es equivalente a $-1 + x_1^2 + x_2^2 = 0 \Rightarrow x_1^2 + x_2^2 = 1$. Representando dicha ecuación en el gráfico, se obtiene la figura 4.9.

Puesto que se ha definido que para $\theta^T x \geqslant 0 \quad y = 1$, las muestras que caigan en dicha recta serán clasificadas cómo clase positiva. Por lo tanto:

$$Clasificar\ y = 1: \quad \theta^T x \geqslant 0 \Leftrightarrow x_1^2 + x_2^2 \geqslant 1$$
$$Clasificar\ y = 0: \quad \theta^T x < 0 \Leftrightarrow x_1^2 + x_2^2 < 1$$

Según las ecuaciones anteriores, se separan dos zonas, la zona de clasificación positiva (zona roja en la figura 4.7) y la zona de clasificación negativa (zona azul en la figura 4.7).

En este ejemplo se ha realizado un límite de decisión más complejo debido al orden de los parámetros empleados para definir la hipótesis. Por otra parte, se pueden realizar límites de decisión todavía más complejos, que presenten unas formas diversas. Se puede dar el caso en el que la clasificación presente una forma más compleja, para hallar los parámetros que describan dicha forma se utilizará una hipótesis tal que:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 \ldots)$$

En la siguiente figura se presentan ejemplos de formas que se pueden llegar a describir utilizando grados mayores de las variables.

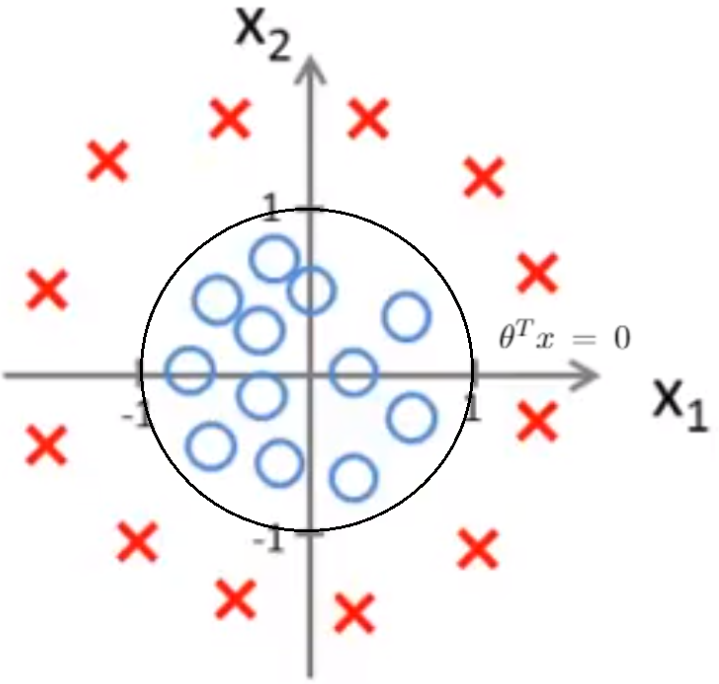


FIGURA 3.9: Límite de decisión

3.3.3. Función de costes

En los ejemplos anteriores se han asumido los parámetros θ de ajuste al modelo. ¿Cómo se hallan los parámetros óptimos para un conjunto de datos de entrenamiento?

Para ello se debe definir cual es el objetivo de la optimización, lo que viene a ser la función de costes. La función de costes representa la suma global de errores cometidos en un conjunto de datos de entrenamiento. Los errores hacen referencia a la diferencia entre el valor predicho y la realidad. La función de costes $J(\theta)$ dependerá de los parámetros θ que se pretenden optimizar para un modelo.

El modelo contará con n características: $x \in [x_0, x_1, x_2, \ldots, x_n]$
El conjunto de datos de entrenamiento contará con m muestras:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

De modo que la función de costes viene dada por:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta^{(i)}, y^{(i)})$$

La función *cost*, que será detallada posteriormente, permite contabilizar el error entre el valor predicho por la hipótesis del modelo utilizado y el valor real. El objetivo de la optimización recae en minimizar $J(\theta)$:

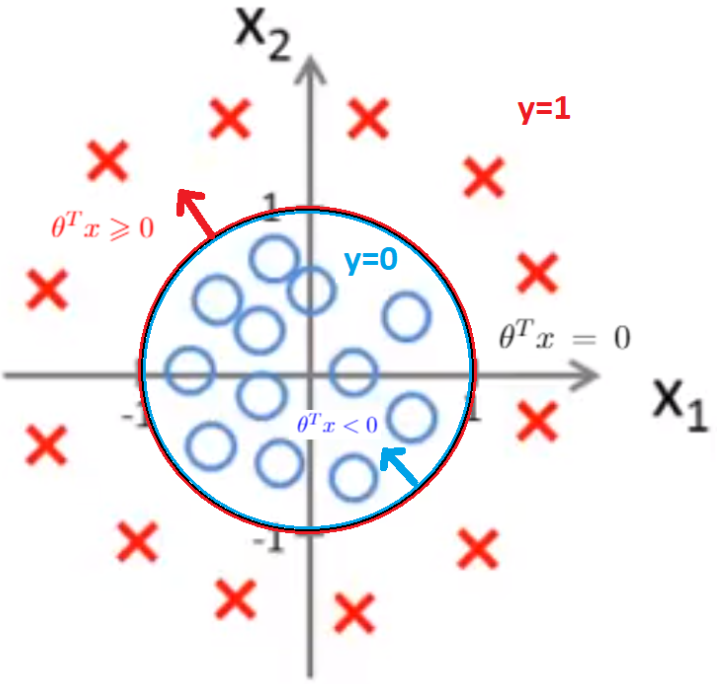


FIGURA 3.10: Zonas de clasificación

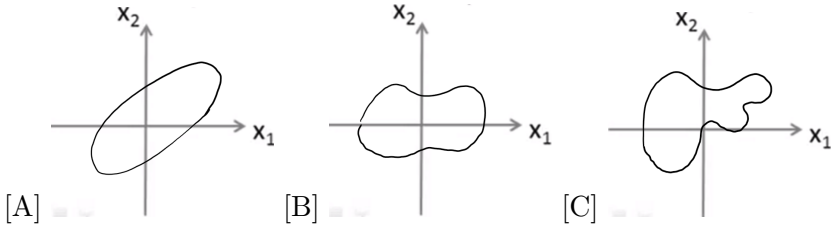


FIGURA 3.11: Límites de decisión complejos

$$\text{mín}_\theta\, J(\theta) \;\Rightarrow\; \text{mín}_\theta\, \tfrac{1}{m} \textstyle\sum_{i=1}^{m} cost(h_\theta^{(i)}, y^{(i)})$$

Para ejecutar dicho problema de optimización, se emplea la técnica de *Gradient Descent* o del descenso de gradiente. Dicha técnica será descrita en el siguiente apartado.
Para que el *Gradient Descent* sea eficiente y encuentre el mínimo global, es necesario que la función *cost* incluida en $J(\theta)$ sea convexa (para más información acerca de los mínimos de una función, consultar Anexo E).

¿Cómo se define la función *cost*?

La función *cost* se separa en dos casos, para valores de *y=0* y valores de *y=1*, de la siguiente manera:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & si \quad y = 1 \\ -log(1 - h_\theta(x)) & si \quad y = 0 \end{cases}$$

El dominio de definición de $Cost(h_\theta(x), y)$ va de *(0,1)* ya que $h_\theta(x)$ sólo puede tomar valores en dicho intervalo.

A continuación se muestran las dos funciones asociadas a *cost*:

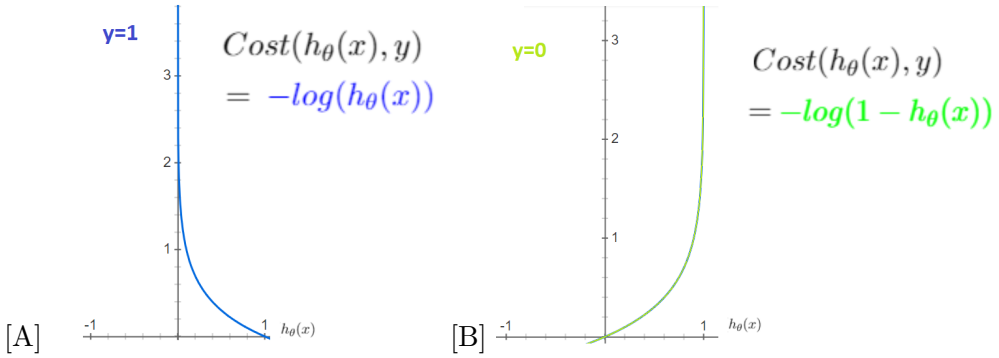


FIGURA 3.12: Función de coste regresión logística

¿Cual es el propósito de dichas funciones? La función logaritmo decimal es muy interesante para la función de costes ya que es convexa. Esta propiedad permite que, a la hora de aplicar *Gradient Descent*, se encuentre el mínimo global. Por lo tanto se podrán encontrar los valores θ que optimizan la función de costes.

Para el caso en que *y=1*, observamos que para valores de $h_\theta(x)$ cercanos a 1, el resultado es prácticamente nulo. Esto implica que si la muestra es de clase positiva y la hipótesis retorna una gran probabilidad de que lo sea, el error será mínimo. En la situación opuesta, donde *y=1* y la hipótesis devuelve probabilidades muy bajas, el error se maximiza, penalizando los parámetros θ vigentes.

En el caso de que *y=o* sucede lo mismo que para *y=1* pero al revés. Si la hipótesis retorna altas probabilidades de que la muestra sea de la clase positiva, el error se maximiza. En cambio, si las probabilidades son bajas, el error es mínimo.

Ambas funciones pueden ser reagrupadas en una única función de costes, de la siguiente manera:

$$Cost(h_\theta(x), y) = -ylog(h_\theta(x)) - (1 - y)log(1 - h_\theta(x))$$

Destacar que cuando *y=1*, entonces el segundo término es nulo (en verde) y no afectará al resultado. Si *y=0*, entonces el primer término es nulo (en azul) y no afectará al resultado.

La función de costes global queda de la siguiente forma:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} -ylog(h_\theta(x)) - (1 - y)log(1 - h_\theta(x))$$

o bien,

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} ylog(h_\theta(x)) + (1-y)log(1-h_\theta(x))$$

Una vez se tiene la función de costes caracterizada, se procede a detallar el método por el cual se minimiza tal función.

3.3.4. Gradient Descent

Gradient Descent es un algoritmo iterativo que minimiza funciones. Dada una función definida por un conjunto de parámetros (*e.g.* θ), *Gradient Descent* comienza con unos valores iniciales para dichos parámetros y de forma iterativa se va desplazando hacia el conjunto de parámetros que minimiza la función. Para hallar un mínimo local de una función, se realizan pasos proporcionales al gradiente de la función en el punto en el que se encuentra. El algoritmo de *Gradient Descent* es el siguiente.

Repeat {
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
}

Los valores de θ tomarán un nuevo valor simultáneamente, hasta que los valores converjan. Esto sucederá cuando la derivada de θ sea prácticamente nula, lo que indicará que para dichos valores de θ, la función de costes se encuentra en su mínimo global. El parámetro α corresponde al parámetro de aprendizaje, se encarga de pronunciar o reducir la variación de los parámetros θ según la derivada de la función de costes (en el Anexo G se detalla el efecto de una α elevada o pequeña a la hora de aplicar *Gradient Descent*).

Para contar con una mejor intuición respecto al algoritmo, el siguiente ejemplo muestra los pasos que realiza el *Gradient Descent* de forma iterativa. En este ejemplo, la función de costes sólo depende del parámetro θ_1, en vistas a simplificar. El algoritmo sería el siguiente:

Repeat {
$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$
}

Si se representa un ejemplo de $J(\theta_1)$ convexo y se inicia el algoritmo con una θ_{ini} :

Se observa en la figura 4.13 que para la θ_{opt} el valor de la función de costes es aproximadamente de 160. Si se inicia el *Gradient Descent* desde la θ_{ini}, se puede apreciar que

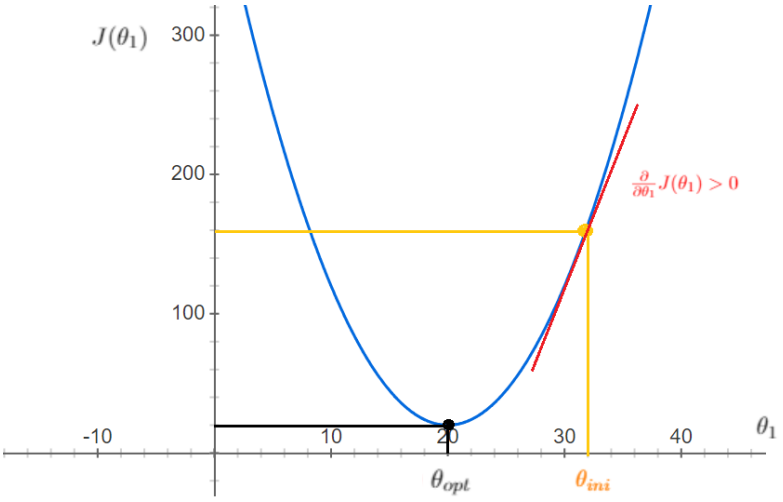


FIGURA 3.13: Aplicación *Gradient Descent* a una función de costes

la función de costes en dicho punto tiene una pendiente positiva. Por lo tanto para la primera iteración, el algoritmo actualizará el valor de θ_1 de la siguiente manera:

$$\theta_{1-iter1} := \theta_{1-ini} - \alpha \frac{\partial}{\partial\theta_1} J(\theta_1) \quad \Leftrightarrow \quad \theta_{1-iter1} := 32 - \alpha \underbrace{\frac{\partial}{\partial\theta_1} J(\theta_1)}_{>0}$$

Por lo tanto, el valor de θ_1 para la primera iteración disminuye y se acerca θ_{opt}. Tras cada iteración el parámetro θ_1 irá decreciendo, hasta que la pendiente sea prácticamente nula. En la figura 4.14 se muestran 3 iteraciones más del algoritmo.

A medida que el algoritmo se acerca al θ_{opt} los pasos son más pequeños, ya que la derivada de la función de costes disminuye.

De esta manera *Gradient Descent* halla los valores θ_{opt} para las funciones de coste. La técnica es la misma cuando existe más de un parámetro θ, en ese caso es de suma importancia actualizar todos los parámetros al mismo tiempo, en cada iteración.

Para el caso de la regresión logística, se sustituye la derivada de θ por la siguiente expresión (ver Anexo F para el detalle):

Repeat {
$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta \left(x^{(i)} \right) - y^{(i)} \right] x_j^{(i)}$
}

Existen otros métodos más rápidos para hallar los parámetros óptimos (*e.g.* "Método del gradiente conjugado", "BFGS" y "L-BFGS"). En este trabajo se trabaja con el algoritmo de *Gradient Descent*.

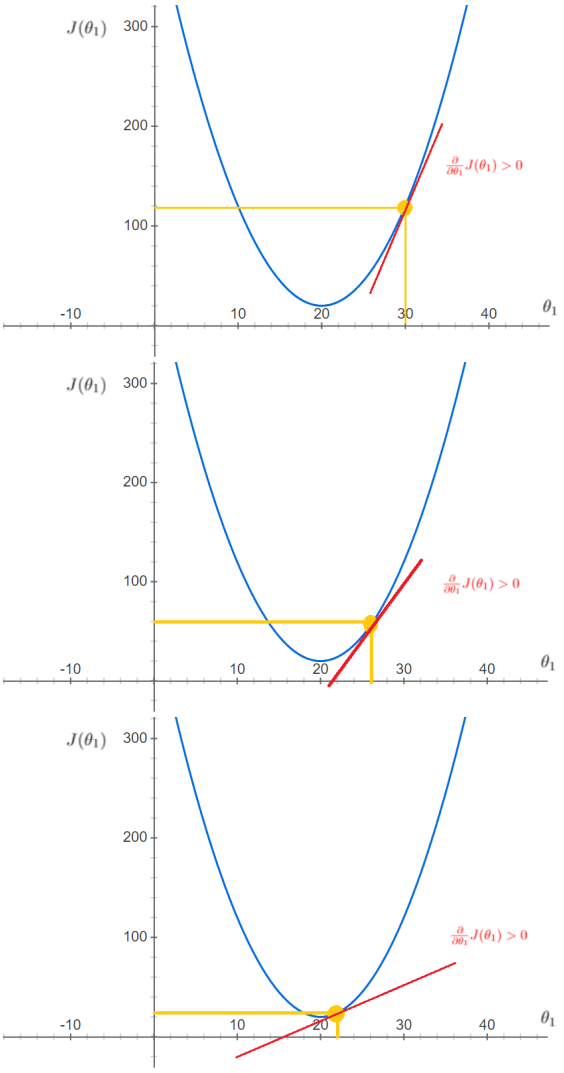


FIGURA 3.14: Iteraciones de *Gradient Descent*

3.4. *Support Vector Machines*

Support Vector Machine (SVM) es un algoritmo de aprendizaje supervisado que puede ser utilizado tanto para problemas de clasificación como de regresión. Este algoritmo traza cada dato como si se tratase de un punto en un espacio n-dimensional, siendo n el nº de parámetros que se tiene y cada uno de estos el valor de una coordenada. Posteriormente, realiza la clasificación buscando el hiperplano que diferencia dos clases.

En su formulación básica *Support Vector Machines* corresponde a un clasificador binario lineal, lo que implica que existe una única frontera de separación entre dos clases, representada por una línea en $\mathbb{R}^2$ y por un hiperplano en el caso general de $\mathbb{R}^n$. Para más información relativa a los hiperplanos, consultar el Anexo A.

Por otra parte, dicho algoritmo puede ser aplicado para clasificación multiclase o para tratar conjuntos no separables linealmente.

3.4.1. Clasificador lineal

En esta sección se comentarán las características esenciales en la formulación básica de los *Support Vector Machines* como clasificadores lineales.

Si se cuenta con dos clases y se quiere definir una frontera lineal entre ellas, no va existir una única solución, existirán diversos hiperplanos de separación válidos.

Realizando el estudio de un ejemplo bidimensional, con dos clases, implica que pueden ser trazadas líneas de separación entre las muestras de la clase 1 y las muestras de la clase 2 haciendo caso a diferentes criterios.

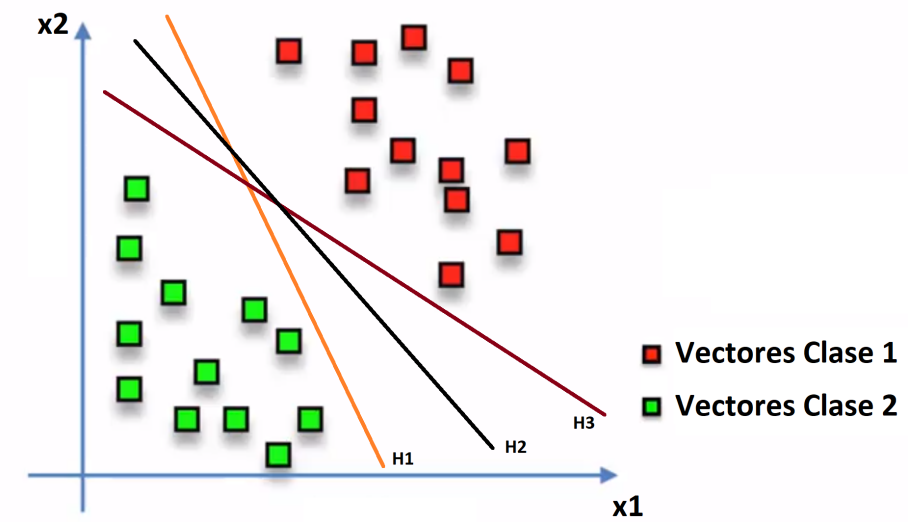


FIGURA 3.15: Ejemplo bidimensional

Dentro de los clasificadores lineales, la característica principal de los *Support Vector Machines* es que la solución se basa en lo que se conoce como margen máximo a partir de los vectores de soporte.

La solución de un *Support Vector Machines* va a ser un hiperplano que particiona el espacio en dos regiones y que para el caso particular de dos dimensiones es una línea recta. Esta solución se hallará a partir de las muestras de entrenamiento.

La particularidad específica del *support vector machine* es que para calcular el hiperplano solución solamente se tienen en cuenta un número limitado de muestras del conjunto de entrenamiento con unas propiedades concretas. A estas muestras de singular importancia se les denomina vectores de soporte, así cada una de las clases tendrá un conjunto de vectores de soporte que en la siguientes figuras corresponderán a los vectores con una cruz en su interior.

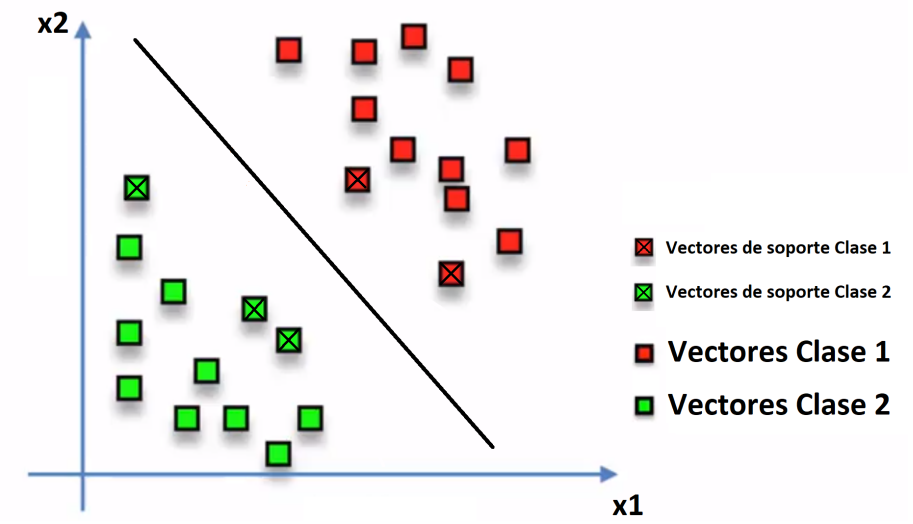


FIGURA 3.16: Vectores de soporte

Los vectores de soporte se eligen de manera que la distancia entre los hiperplanos de vectores de soporte para cada clase sea máxima. Esta distancia es lo que se conoce como margen, el objetivo de los SVM reside en maximizarlo (para conjuntos linealmente separables, se denomina hard margin o margen rígido). Según un punto de vista matemático, para lograr el objetivo se debe plantear un problema de optimización en el cual la función a optimizar es precisamente el margen.

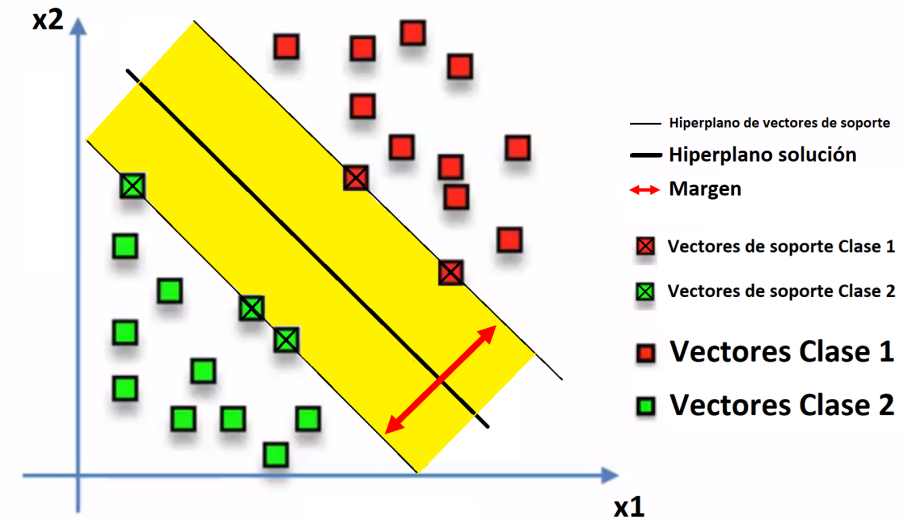


FIGURA 3.17: Hiperplano solución óptimo

Esta condición de margen máximo implica que se halla la región más amplia de el espacio de características que separa las dos clases y que además esta se encuentre vacía de muestras.

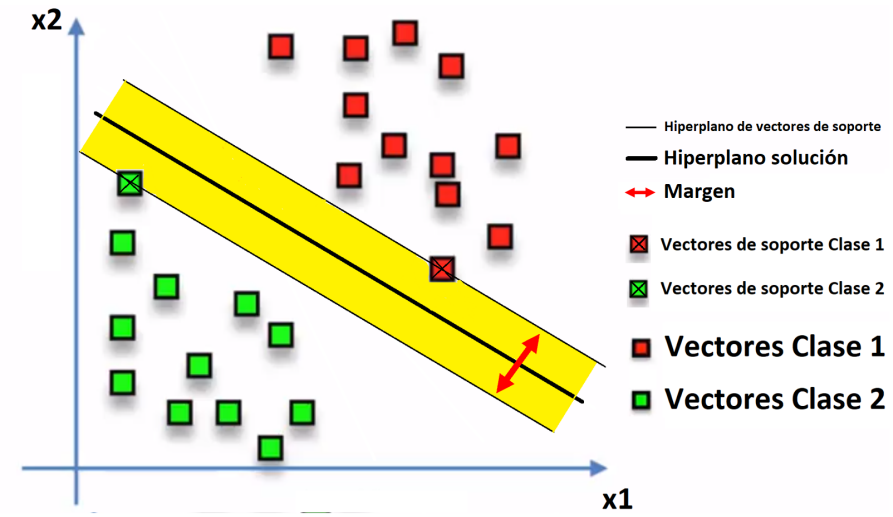


FIGURA 3.18: Hiperplano solución posible

El hiperplano que se encuentra equidistante a los hiperplanos de vectores de soporte se trata de la solución del support vector machine.

Por esta razón, cualquier elección alternativa de los vectores de soporte (Figura 4.1, H1 Y H3) y por lo tanto cualquier solución alternativa al hiperplano de separación entre clases, dará como resultado un margen más estrecho y por lo tanto no va a ser una solución óptima ya que se tendría un margen de seguridad menor en la clasificación de muestras que estén próximas a las fronteras entre clases.

A la hora de usar los vectores de soporte debemos tener en cuenta lo siguiente:

- Si se desplazan las muestras que son vectores de soporte, resultará en la modificación de la solución, puesto que son dichos vectores los que definen el margen.Se puede observar en la Figura 4.5 [A] que, debido a que los vectores de soporte se han acercado, el margen de seguridad es menor.
- Debido a que las muestras que no son vectores de soporte no intervienen en la definición de la frontera de separación, si se desplaza cualquier muestra que no sea un vector de soporte, la solución permanecerá inalterada. Como se puede observar en la Figura 4.5 [B], los vectores que no son de soporte tienen una nueva configuración y no por ello el hiperplano se ha modificado. Esto proporciona cierta robustez estadística, particularmente frente a lo que se conoce como *overfitting*. El *overfitting* (que se explicará en detalle más adelante) implicaría perder capacidad de generalización por ajustarse demasiado a las muestras de entrenamiento.

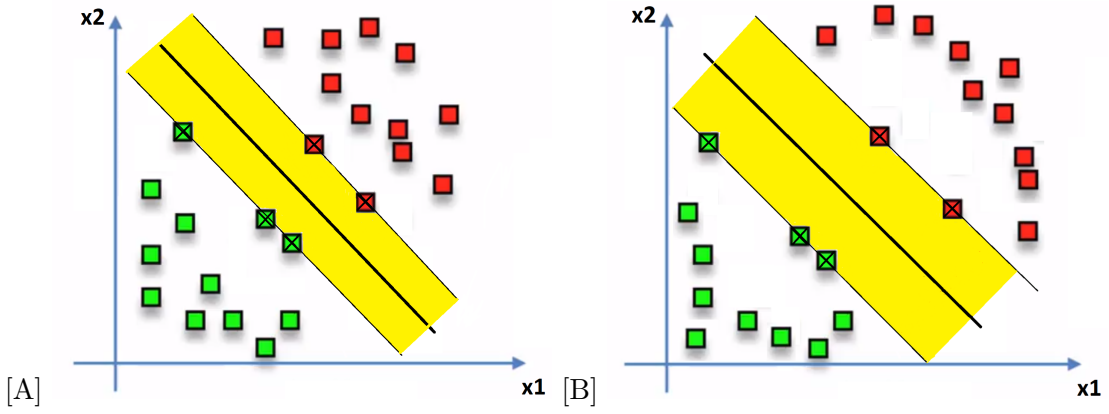


FIGURA 3.19: Propiedades Vectores de Soporte

3.4.2. Clasificador de conjuntos linealmente no separables

El modelo que se ha tratado en el apartado anterior no suele ser representativo de los conjuntos de datos que representan la realidad. Los conjuntos reales suelen presentar solapamiento entre las clases ya que no será posible realizar de forma lineal la separación de clases. Como se puede observar en la Figura 4.7, que representa un conjunto de datos más habitual, no existe un hiperplano lineal que separe las dos clases.

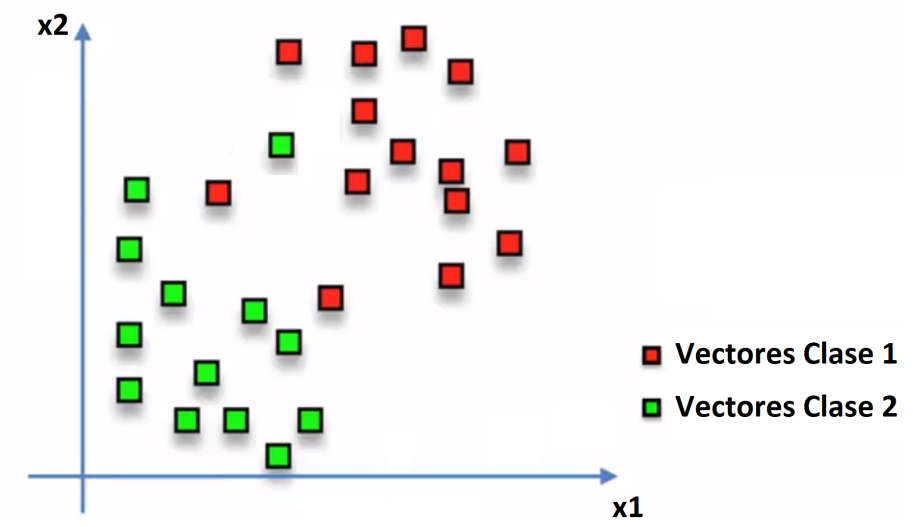


FIGURA 3.20: Conjunto de datos habitual

Ante tal escenario, para que el support vector machine pueda ser eficiente existen dos alternativas: relajar la condición de margen y el *kernel trick*. Por un lado se permite un cierto solapamiento entre clases a partir de relajar la condición de margen máximo. Por otro lado se puede extender el modelo a conjuntos que no sean linealmente separables mediante la transformación del espacio de características original en otro espacio que sí sea linealmente separable, aplicando el *kernel trick*.

3.4.2.1. Relajar la condición de margen

La relajación de la condición del margen o *soft margin* (margen suave) se utilizará en el caso de que a pesar de existir posibles sesgos, una separación lineal tenga sentido. Idealmente, el modelo basado en SVM debería producir un hiperplano que separe completamente los datos del universo estudiado en dos categorías. Sin embargo, una separación perfecta no siempre es posible y, si lo es, el resultado del modelo no puede ser generalizado para otros datos. Esto se conoce como *overfitting* (Anexo B).

Con el fin de permitir cierta flexibilidad, los SVM manejan un parámetro C que controla la compensación entre errores de entrenamiento y los márgenes rígidos, creando así un *soft margin* que permita algunos errores en la clasificación a la vez que los penaliza. Esta relajación implica que sí va a ser posible que algunas muestras caigan dentro de la zona del margen, o dicho de otra manera, que queden erróneamente clasificadas.

Cuando el parámetro C es pequeño, el clasificador permite únicamente pocos errores de clasificación. De esta manera se reducen los sesgos pero por contra puede no generalizar bien para nuevas muestras. Para estos valores del parámetro C se puede generar *overfitting*. Si C es elevado, el clasificador generalizará mejor, pero tendrá un sesgo mayor. La transición que observamos en la figura 4.8 a nivel del margen pasa por un parámetro C nulo (margen rígido) en la figura A y más elevado en la B (margen suave).

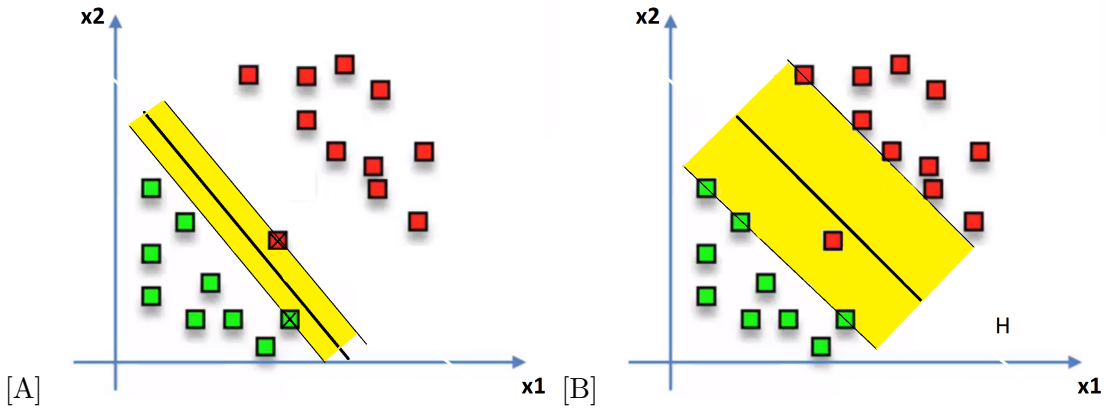


FIGURA 3.21: Margen rígido y suave

En resumidas cuentas, el margen suave hará que el SVM pueda ser robusto al ruido asociado a las muestras. Por otra parte, cuenta con una cierta tolerancia a errores, de manera que se puede gestionar el problema de solapamiento de clases además de permitir trabajar con conjuntos no separables.

3.4.2.2. *Kernel trick*

Desafortunadamente los universos a estudiar no se suelen presentar casos idílicos de dos dimensiones como en los ejemplos anteriores, sino que un algoritmo SVM debe tratar con:

- Más de dos variables predictoras.
- Curvas no lineales de separación.
- Casos donde los conjuntos de datos no pueden ser completamente separados.
- Baja eficiencia de un clasificador lineal con relajación de margen.

Debido a la complejidad de los conjuntos de datos, las máquinas de aprendizaje lineal no pueden ser utilizadas en la mayoría de las aplicaciones del mundo real. En la figura 4.9 se aprecia dicho problema.

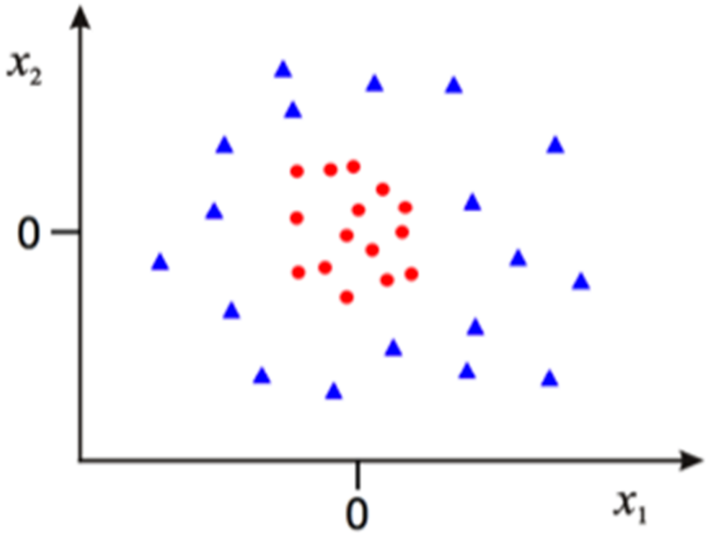


FIGURA 3.22: Conjunto linealmente no separable

La representación por medio de funciones Kernel ofrece una solución a este problema. Para ello se *mapea* (traslada y representa) el conjunto de características desde el espacio original en el que los datos no son linealmente separables a un espacio de dimensión superior en el que los datos son linealmente separables. Es decir, se *mapea* el espacio de entradas X a un nuevo espacio de características de mayor dimensionalidad. La siguiente ecuación representa el *mapeo* de las características a un espacio de mayor dimensionalidad:

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

De esta manera, en el nuevo espacio tridimensional, los datos quedan de la siguiente forma:

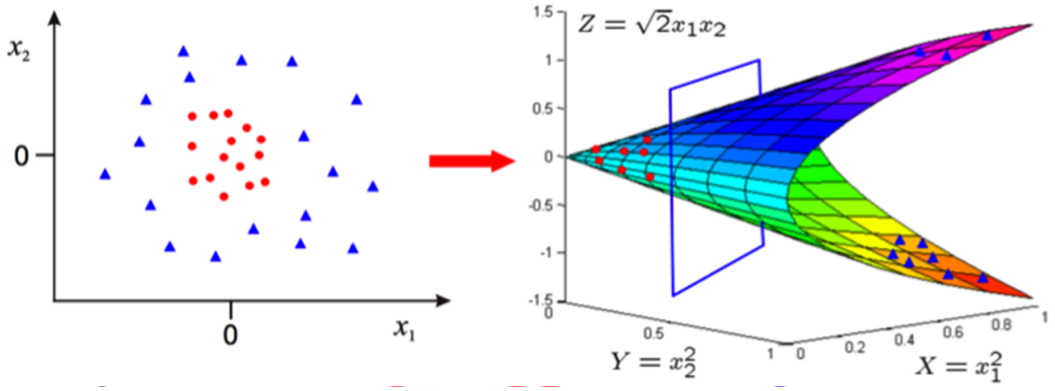


FIGURA 3.23: Aplicación del kernel trick

Se observa que gracias a esta transformación, el nuevo espacio del problema pasa a ser linealmente separable y que por lo tanto se puede resolver utilizando *Support Vector Machines* de forma estándar. La función matemática utilizada para la transformación se conoce como función *kernel*.

Existen diferentes *kernels*, entre ellos están los *Polynomial kernel* y *Radial basis function kernel* (RBF).

3.5. *Dataset*

El objetivo de este proyecto trata de crear un modelo que prediga si la fabricación de un equipo industrial será la correcta o, en su defecto, si será necesario el reajuste de los equipos. Para ello es necesario contar con un *Dataset*, o conjunto de datos, para poder entrenar un modelo utilizando (en este caso) los algoritmos de regresión logística y *Support Vector Machines*.

Se ha buscado en las redes *Datasets* propios de equipos industriales y no ha sido posible hallar ningún *Dataset* abierto al público.

Debido a dicho contratiempo, se ha decidido utilizar un *Dataset* accesible y con unas características similares a las que se pretendía analizar en un inicio. El *Dataset* presenta un gran número de transacciones realizadas con tarjetas de crédito. Estas transacciones

se caracterizan por 28 variables distintas, además de la cantidad y el instante en el que se efectuaron. Las transacciones reciben la etiqueta de "Transacción fraudulenta"(*y=1*) o "Transacción no fraudulenta"(*y=0*) según sea el caso.

El objetivo para dicho *Dataset* es el de minimizar los errores cometidos por el clasificador a la hora de indicar si una transacción es fraudulenta o no. Al tratarse de un clasificador binario, cada una de las predicciones se puede encontrar en uno de los siguientes escenarios:

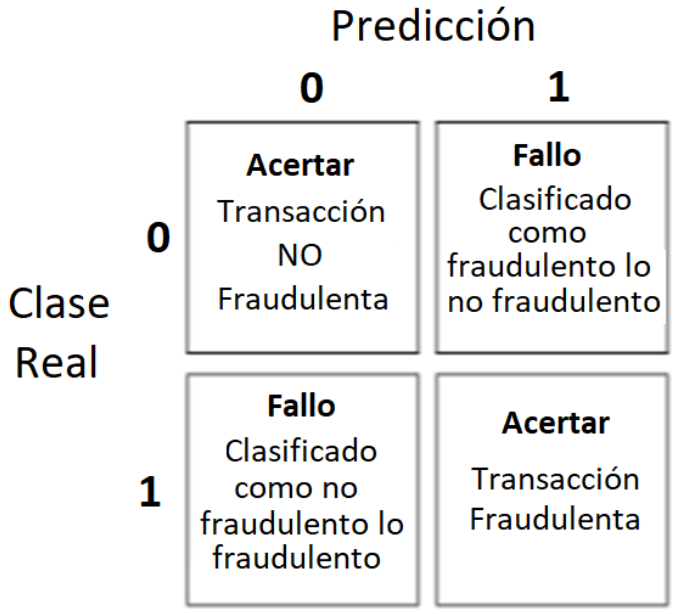


FIGURA 3.24: Errores en el contraste

Existen dos tipos de fallos a la hora de predecir:

- Cometer el error de clasificar cómo clase positiva a una clase negativa. La probabilidad de que esto suceda se conoce como error tipo I o bien α.

$$P(escoger H_1 | H_0 \; es \; cierta) = \alpha$$

- Cometer el error de clasificar cómo clase negativa a una clase positiva. La probabilidad de que esto suceda se conoce como error tipo II o bien β.

$$P(escoger H_0 | H_1 \; es \; cierta) = \beta$$

Tras aplicar los distintos clasificadores, a los conjuntos de datos de prueba, se obtendrán tablas como la de la figura 4.24, indicando el número de aciertos y fallos.

3.5.1. Características del *dataset*

El *dataset* proviene de la página web Kaggle. Kaggle es una plataforma para el modelado predictivo y competencias de análisis en el que se pueden encontrar una gran variedad de conjuntos de datos abiertos al público. Kaggle recuperó dicho *dataset* de una colaboración de investigación entre Wordline y The Machine Learning Group en la universidad de Bruselas (ULB). Dicha investigación era acerca de la exploración de grandes bases de datos y la detección de fraude.

El *dataset* contiene transacciones realizadas mediante tarjeta de crédito, en septiembre de 2013 por usuarios europeos. En concreto este conjunto de datos presenta las transacciones que sucedieron en dos días. En este intervalo de tiempo sucedieron 492 fraudes de un global de 284.807 transacciones. Se aprecia que el conjunto de datos está claramente desequilibrado, ya que dentro de la totalidad de transacciones, un 0,172 % son fraudes.

Las variables que caracterizan las transacciones son únicamente numéricas, estás han sido transformadas mediante Análisis de Componentes Principales (ver siguiente apartado). A excepción del tiempo "Time", de la cantidad de la transacción "Amount" y la clase "Class" que no han sido transformadas mediante ACP. La variable del tiempo contiene los segundos que existen entre una transacción y la siguiente. La variable "Class" es la variable respuesta, toma el valor de 1 para los fraudes y 0 para las transacciones no fraudulentas. Debido a ciertas cuestiones de confidencialidad, no se cuentan con las variables originales ni con información acerca del conjunto de datos.

3.5.1.1. Análisis de componentes principales

El Análisis de Componentes Principales (ACP) es una técnica utilizada para describir un conjunto de datos en términos de nuevas variables ("componentes principales") no correlacionadas. Generalmente el ACP describe el conjunto de datos con un número menor de variables, estas variables se conocen como componentes principales (CP).

Cuando se recoge la información de una muestra de datos, lo más frecuente es tomar el mayor número posible de variables. Sin embargo, si se toman demasiadas variables sobre un conjunto de objetos, por ejemplo 20 variables, tendremos que considerar $\binom{20}{2} = 190$ posibles coeficientes de correlación; si son 30 variables dicho número aumenta hasta 835. Con tantas combinaciones es difícil visualizar relaciones entre las variables. Otro problema que se presenta es la fuerte correlación que muchas veces se presenta entre las variables: si se toman demasiadas variables, lo normal es que estén relacionadas o que midan lo mismo bajo distintos puntos de vista. Por ejemplo, en un equipo industrial, el desgaste de una herramienta de corte y la profundidad de corte en las piezas procesadas

estén fuertemente relacionadas. Se hace necesario, pues, reducir el número de variables. Es importante resaltar el hecho de que el concepto de mayor información se relaciona con el de mayor variabilidad o varianza. Cuanto mayor sea la variabilidad de los datos (varianza) se considera que existe mayor información.

El ACP trata de presentar de una forma más clara y ordenada la diversidad de la información. Para ello buscará un nuevo sistema de ejes con el que poder apreciar y analizar más claramente la diversidad de comportamiento reflejada en los datos. Determinará como primer eje coordenado la nueva variable (primera componente principal) que explique la máxima variabilidad posible de los datos observados, para proceder secuencialmente y de forma análoga a determinar los sucesivos ejes coordenados a partir del resto de la variabilidad de los datos, aún no explicada por los anteriores.

Las variables originales x son transformadas a CP. En la siguiente figura se aprecia que las componentes principales reflejan la mayor varianza.

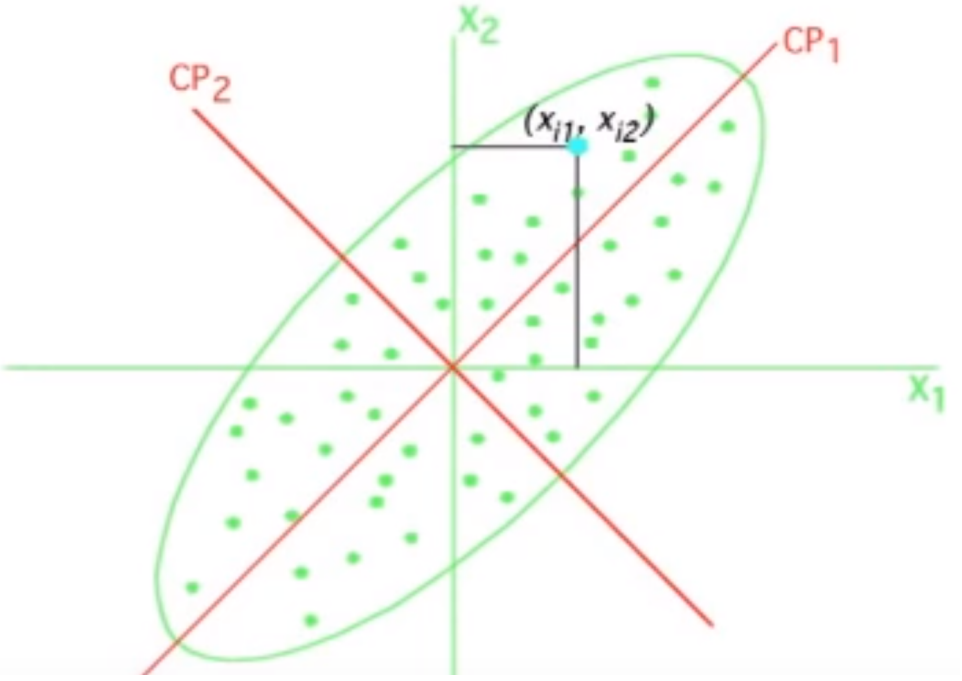


FIGURA 3.25: ACP bidimensional

3.5.2. Exploración del dataset

Es básico a la hora de tratar un conjunto de datos, comprender sus propiedades. Las siguientes figuras se encuentran en el Anexo H.

En primer lugar se ha verificado que no existieran valores nulos en el conjunto de datos. Efectivamente, no existe ningún vacío en el *dataset*. Por lo tanto, tendremos en cuenta la totalidad del conjunto de datos, para confeccionar los modelos más adelante.

De las 284807 transacciones, se presenta el siguiente escenario:

- 492 transacciones fraudulentas, lo que corresponde a un 0,17 % (aproximadamente un 0,2 % de las transacciones son fraudulentas). Estas transacciones tienen un valor global de 60127,97 U.M.
- 284315 transacciones no fraudulentas, lo que corresponde a un 99,83 %. Estas transacciones tienen un valor global de 25102462,04 U.M.

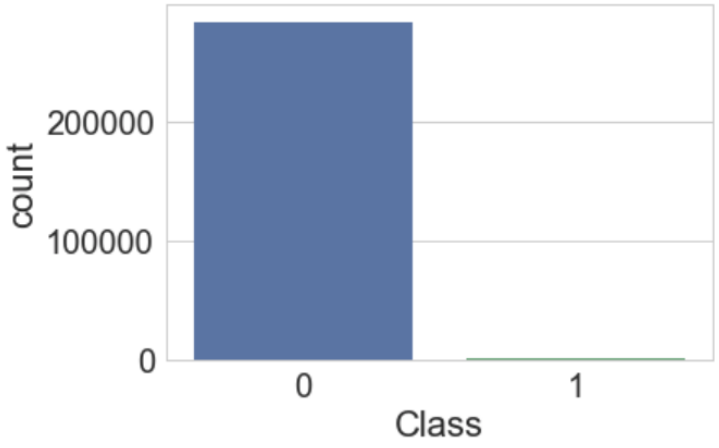


FIGURA 3.26: Dataset desequilibrado

Se puede apreciar que el conjunto de datos está altamente desequilibrado. Este dato es muy relevante para la confección posterior de los modelos. Tendremos que utilizar técnicas para evitar el problema de *overfitting*, estas serán detalladas en el capítulo 5.

Si se analiza en detalle los casos de fraude y se extrae la relación cantidad en función del tiempo, se obtiene el siguiente gráfico:

En primer lugar, destacar que el tiempo no tiene ninguna consecuencia en la suma defraudada. En segundo lugar, la suma defraudada presenta una gran variabilidad, comenzando desde las 0UM [2] a las 2125.87UM. Por último, la media de las sumas defraudadas no es muy elevada. De promedio se defraudan 122,21UM.

Por otra parte, se realiza un estudio de la correlación de las variables para verificar que se ha realizado previamente el análisis de componentes principales correctamente.

[2] Unidades Monetarias

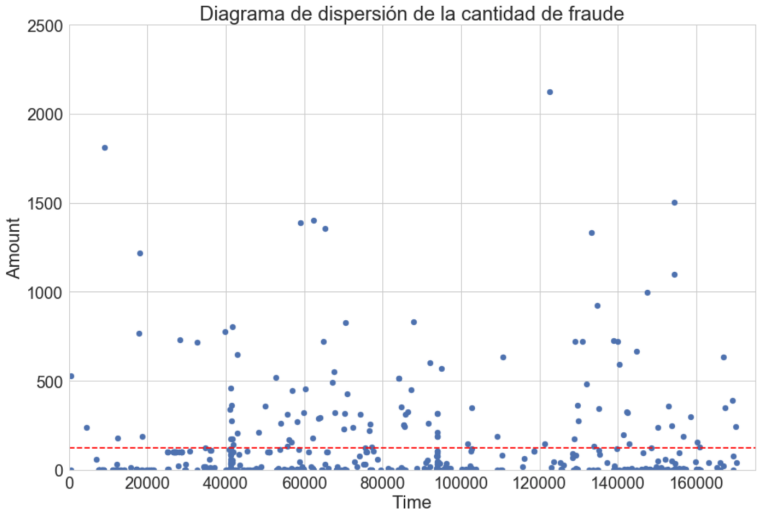


FIGURA 3.27: Mapa de correlación

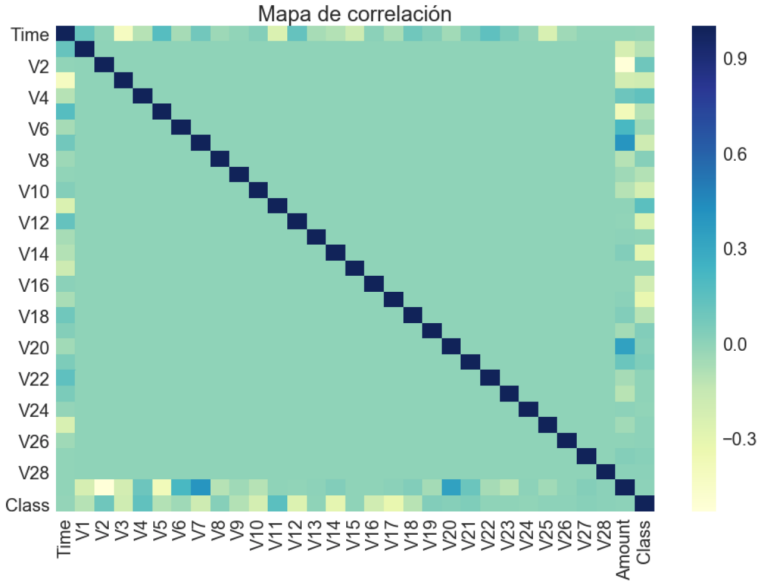


FIGURA 3.28: Mapa de correlación

En el mapa de correlación se observa que no existe correlación entre variables, lo que implica que el ACP ya venía aplicado en el *dataset*.

Por último, cabe destacar que en el caso de que un clasificador realizara siempre la predicción de la clase negativa sobre el conjunto de datos, se obtendría el siguiente resultado:

En la matriz de confusión (figura análoga a la figura 4.24), se puede destacar que la precisión es muy elevada ya que la gran mayoría de transacciones son no fraudulentas. De hecho la precisión sería del 99,83 %. Lo que sucede es que dicho clasificador no responde al objetivo fijado, el cual busca minimizar el error tipo II. En esta situación el error tipo II sería igual a 1, puesto que el clasificador siempre se equivoca.

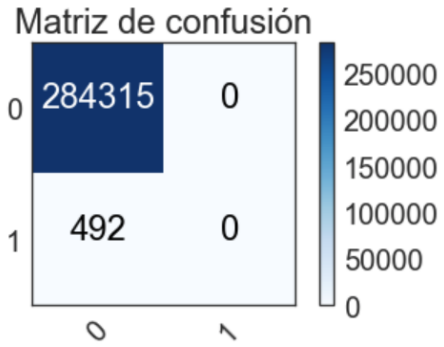


FIGURA 3.29: Matriz de confusión: clasificador clase negativa

En el capítulo siguiente se observará que para reducir el error tipo II, el error tipo I aumentará. Debido a este fenómeno, se buscará un equilibrio para ambos errores, en el que ambos tengan los valores más pequeños posibles.

Capítulo 4

Análisis y comparación de resultados

En este capítulo, se va a proceder a analizar los resultados obtenidos con distintos modelos estadísticos de *Machine Learning*. En un primer tiempo, el análisis se centrará en valorar el rendimiento de la regresión logística en la tarea de reconocimiento de patrones en los datos de fraude bancario ya presentados anteriormente. A continuación, se estudiará la capacidad predictiva de los SVM (*Support Vector Machines*), variando distintos hiperparametros de interés. Finalmente, se realizará una comparativa entre estos dos modelos predictivos lineales, mencionando en qué escenarios resultarían más adecuados.

4.1. Clasificación mediante regresión logística

4.1.1. Librerías

Se han utilizado diversas librerías, en el entorno de *Python*, para poder manipular el *dataset* de Kaggle y crear modelos estadísticos.

En primer lugar se ha importado la librería *pandas*. Esta resulta muy útil para operar con *datasets*, ya que permite manipular tablas numéricas y series temporales. Se trata de una librería de software para la manipulación y análisis de datos.

En segundo lugar se ha importado la librería *numpy*. *Numpy* es una librería con funciones matemáticas de alto nivel, ofrece un mayor soporte para operar con vectores y matrices. Se realizarán operaciones con esta librería, con la información extraída mediante pandas.

En tercer lugar se ha importado la librería *Matplotlib*. *Visualization gives you answers to questions you didn't know you had* Ben Shneiderman (La visualización responde a preguntas que no sabias que tenías). *Matplotlib* responde a la necesidad que el informático Ben Schneiderman presenta, visualizar la información para extraer nuevas ideas. *Matplotlib* se trata de una librería para la generación de gráficos a partir de datos. Las listas de datos generadas a través de numpy se podrán visualización gráficamente gracias a *Matplotlib*.

Por último la librería Scikit learn se ha importado para entrenar los modelos de *Machine learning*. Scikit learn es una librería que contiene diversos algoritmos de *Machine learning*. En el marco de este proyecto, mediante esta biblioteca se entrenan los modelos de regresión logística y SVM.

4.1.2. Preproceso del *dataset*

En este apartado se detallan las manipulaciones que se han realizado al *dataset* para su posterior análisis.

En la exploración del *dataset* realizada en el capítulo 3, se determina que el *dataset* se encuentra desequilibrado puesto que existe una amplia mayoría de muestras de la clase 0 (transacciones no fraudulentas). Dicho desequilibrio se puede apreciar en la figura 3.25. De manera que este desequilibrio no influya negativamente en el entrenamiento de los modelos, se proponen dos soluciones en el apartado 4.1.2.2.

Por otra parte, la variables "Time" (tiempo) corresponde al tiempo transcurrido entre cada transacción. Este dato no tiene ninguna relevancia en el análisis puesto que las transacciones se realizan en un marco europeo (donde existen distintas franjas horarias) y relaciona dos transacciones sin vínculo aparente. Es por ello que se ha tomado la decisión de suprimirla.

Respecto a las demás variables, las variables V1 a V28 se encuentran estandarizadas, resultado del análisis principal de componentes realizado de antemano sobre el *dataset*. Es necesario estandarizar todas las variables para poder comparar datos procedentes de diferentes muestras o poblaciones. Al estandarizar se obtienen variables con una media nula y una desviación típica unitaria. No es el caso de la variable "Amount" (cantidad), que representa la cantidad exacta en unidades monetarias. Se ha estandarizado la variable "Amount" de manera que la relevancia de dicha variable en los modelos predictivos no se vea afectada frente a las demás variables. Se debe evitar que por la dimensión de los valores que puede tomar la cantidad (elevados) frente al de las otras variables (todas acotadas entre 0 y 1) provoque alteraciones en el modelo.

4.1.2.1. Definición del *training, validation* y *test set*

Para la elaboración de distintos modelos de clasificación, necesitamos separar nuestro *dataset* en tres conjuntos de datos: *training set*, *validation set* y *test set*.

El *training set*, o conjunto de entrenamiento, corresponde al conjunto de datos empleados para entrenar los modelos. Los parámetros, que caracterizan los modelos de clasificación, se determinarán a través de la aplicación de algoritmos sobre dicho conjunto de datos. La regresión logística se trata de un algoritmo supervisado, por lo que requiere de dicho conjunto de entrenamiento para ajustar sus parámetros.

El *validation set*, o conjunto de validación, se utiliza para probar los diferentes modelos que se hayan entrenado con el objetivo de conocer sus rendimientos. Una vez los modelos entrenados con el *training set*, se realiza la predicción con los inputs (entradas) del *validation set* y se compara con los outputs (salidas) del mismo conjunto. Mediante la comparación de eficiencias de los diferentes modelos sobre el *validation set*, se escoge el modelo considerado como el óptimo.

El *test set*, o conjunto de prueba, se trata del conjunto de datos que permite estimar la eficiencia del modelo escogido. Este conjunto es completamente nuevo para el modelo puesto que no ha sido utilizado ni para el entrenamiento ni para la validación. De esta manera se puede evaluar el poder predictivo real del modelo.

El *dataset* de Kaggle presenta un total de 284.807 transacciones, se cuenta con un conjunto relativamente grande. Se separa el *dataset* en tres conjuntos, en la siguiente figura se muestra la división:

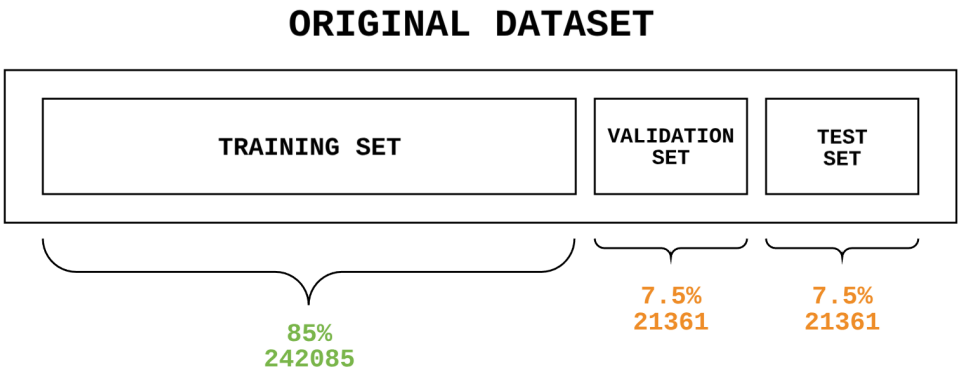


FIGURA 4.1: División del *dataset*

Para decantarse por los porcentajes que figuran en la figura , se ha destacado el hecho de que varias decenas de miles de muestras (21.361) es ampliamente suficiente para validar y probar nuestros modelos. El 85 % de muestras restantes son aprovechadas para entrenar los modelos.

Es importante destacar que para realizar la comparativa entre los modelos obtenidos por regresión logística y SVM, se emplean los mismos conjuntos de datos (training, validation y test).

4.1.2.2. Equilibrar el *dataset*

¿Cómo resolver el desequilibrio presente en el *dataset*?

El problema de los *datasets* desequilibrados reside en que los modelos sesgarán la predicción hacia la clase más común. En el caso de las transacciones fraudulentas, si se tratan los datos de forma desequilibrada, la tendencia del modelo será de clasificar las transacciones como no fraudulentas.

Con el objetivo de evitar dicha tendencia, se aplican las técnicas de *oversampling* y *undersampling* que permiten equilibrar el *dataset*. Ambas técnicas se han aplicado al *training set* para conocer su efecto en el entrenamiento de los modelos.

El *training set* presenta 423 transacciones fraudulentas y 241.662 no fraudulentas, no cabe duda de que se encuentra desequilibrado. A continuación se aplican las dos técnicas de equilibrado mencionadas anteriormente.

- ***undersampling.***
  El *undersampling*, o submuestreo, consiste en reducir el número de muestras de la clase dominante, de un *dataset*, al número de muestras de la clase minoritaria. La clase minoritaria corresponde a las transacciones fraudulentas. Se debe entonces extraer aleatoriamente 423 transacciones no fraudulentas de las 241.662 presentes en el *training set*. De esta manera se cuenta con un *training set* de submuestreo, con un total de 846 muestras (423 fraudulentas/423 no fraudulentas).

- ***oversampling.***
  El *oversampling*, o sobremuestreo, consiste en aumentar el número de muestras de la clase minoritaria, de un *dataset*, al número de muestras de la clase mayoritaria. Corresponde a la técnica inversa al *undersampling*. La clase mayoritaria corresponde a las transacciones no fraudulentas. Se debe entonces añadir replicas aleatorias de las muestras fraudulentas, hasta alcanzar el número de transacciones no fraudulentas. De esta manera se cuenta con un *training set* de sobremuestreo, con un total de 483.324 muestras (241.662 fraudulentas/ 241.662 no fraudulentas).

Se entrenarán dos modelos a partir de un *training set* de submuestreo y otro de sobremuestreo, con el objetivo de analizar su eficiencia. Por otra parte, también se entrenará un modelo con el *training set* desequilibrado, para poder comparar los tres modelos.

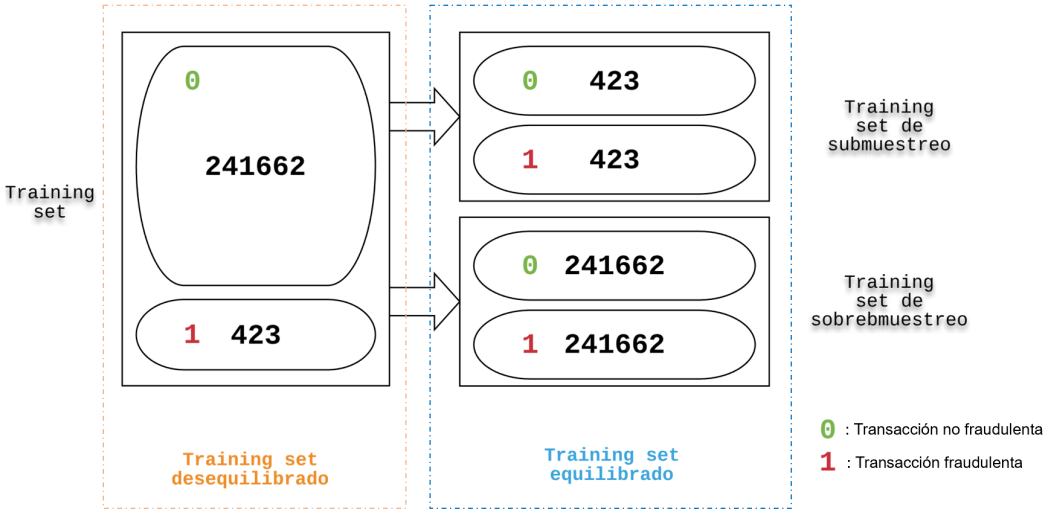


FIGURA 4.2: Tres opciones de *training set*

4.1.3. Métricas

Previo a la presentación de las métricas empleadas, se debe destacar que una manera visual de presentar los resultados es la matriz de confusión (ver figura 3.24). En esta matriz se presentan los aciertos y fallos cometidos por un clasificador, a la hora de predecir.

Se busca minimizar el error tipo 2, se debe evitar clasificar de no fraudulento a las transacciones fraudulentas. Por otra parte, tampoco se debe descuidar el error tipo 1, que debe ser lo más reducido posible.

Para valorar la eficiencia de los modelos, se emplearán las siguientes métricas: precisión, exactitud, exhaustividad, F1-score, ROC AUC score.

Tomaremos la siguiente notación para describir las métricas: TP (acertar transacción fraudulenta), TN (acertar transacción no fraudulente), FN (clasificar como no fraudulento un transacción fraudulenta), FP (clasificar fraudulenta una transacción no fraudulenta).

La precisión es la probabilidad de no predecir que una clase es positiva cuando es negativa.

$$Precisión = \frac{TP}{TP+FP}$$

La exactitud hace referencia al número de predicciones acertadas. Esta métrica no es interesante ya que sin crear ningún modelo, utilizando un clasificador que siempre predijera que las transacciones no son fraudulentas, tendría una exactitud muy elevada.

$$Exactitud = \frac{TP+TN}{TP+TN+FP+FN}$$

La exhaustividad es la probabilidad de predecir las clases positivas. Esta métrica resulta especialmente interesante ya que una alta exhaustividad implica un error tipo 2 reducido.

$$Exhaustividad = \frac{TP}{TP+FP}$$

Generalmente, para evaluar la eficiencia de un clasificador, se emplea el F1 score. Esta métrica tiene en cuenta tanto la exhaustividad como la precisión.

$$F1score = 2\frac{Precision*Exhaustividad}{Precision*Exhaustividad}$$

Debido a la propiedad desequilibrada de nuestro *dataset*, se le dará más importancia a la métrica ROC AUC. La ventaja de esta métrica es que es sensible a los desequilibrios, aportando peso a los errores cometidos durante la clasificación de clases minoritarias.

Para comprender el ROC AUC score se presenta el espacio ROC a continuación:

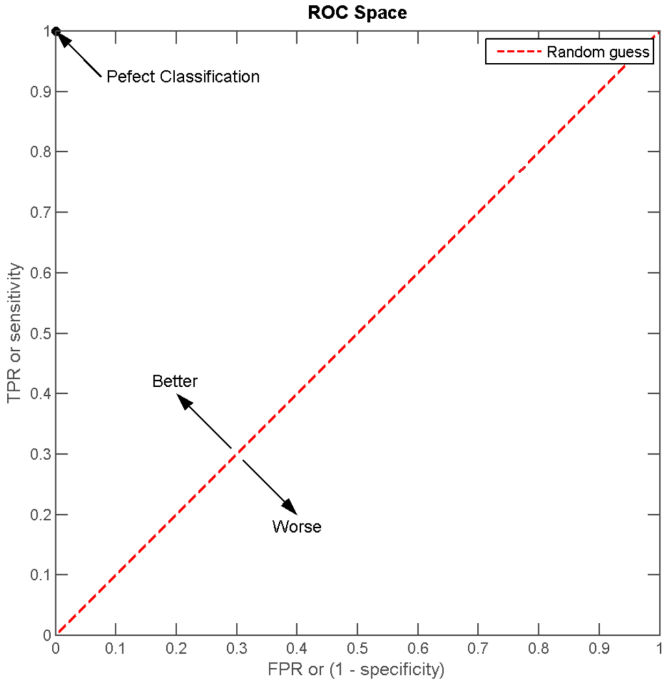


FIGURA 4.3: Espacio ROC

Se presenta la tasa de acierto de clases positivas (acierto de transacciones fraudulentas) en función de la tasa de falsos positivos (predecir fraudulento cuando no lo es). Si la curva se encuentra por encima de la diagonal, representa que el modelo predice mejor que un modelo que clasifica aleatoriamente. Los modelos de clasificación suelen presentar una curva que aumenta medida que los falsos positivos aumentan (varios ejemplos en la figura)

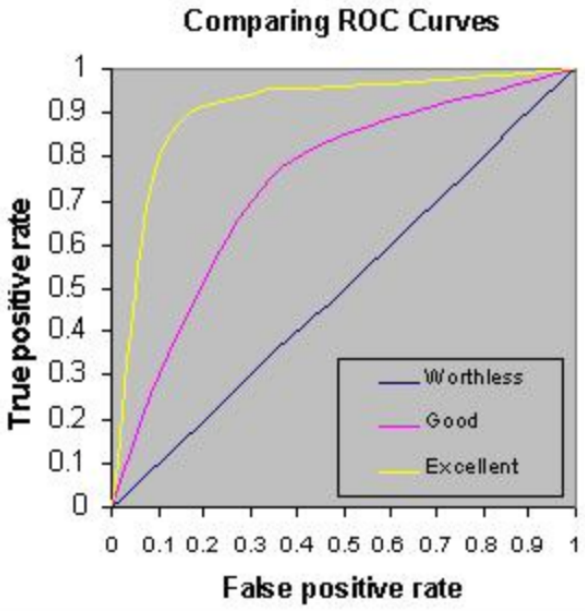


FIGURA 4.4: Curvas ROC

Cuanto mayor sea el área que se encuentra debajo de la curva, mejor clasificador será. El valor del área corresponde al valor de la métrica ROC AUC. Según los resultados, los modelos se pueden clasificar en:

ROC AUC	Calidad del clasificador	Puntuación
,90-1	Excellent	A
,80-,90	Good	B
,70-,80	Fair	C
,60-,70	Poor	D
,50-,60	Fail	F

CUADRO 4.1: Métrica ROC AUC

Presentar un valor inferior a 0,5 no es posible puesto que implicaría que el modelo predice peor que un modelo aleatorio.

4.1.4. Modelos estadísticos

A continuación se detallan los resultados obtenidos al predecir el conjunto de datos de validación de los diferentes modelos.

4.1.4.1. *undersampling* y *oversampling*

Los resultados que presentan los modelos de regresión logística entrenados con el train set desequilibrado, train set de submuestreo y train set de sobremuestreo aparecen en el

cuadro 4.2.

Métrica	**Train set desequilibrado**	***undersampling***	***oversampling***
Exactitud	0,99944	0,96989	0,980946
Precisión	0,9	0,05037	0,07745
Exhaustividad	0,75	0,94444	0,94444
F1 score	0,81818	0,09564	0,14316
ROC AUC	0,87493	0,95719	**0,96273**

CUADRO 4.2: Resultados *undersampling* y *oversampling*

La gran diferencia entre un entreno desequilibrado o equilibrado reside en que por una parte la precisión es mayor en el desequilibrio pero la exhaustividad es menor. En un *dataset* desequilibrado, la exhaustividad es fundamental.

Debido a que el F1 score tiene en cuenta la precisión, los modelos entrenados con *undersampling* y *oversampling* tienen un valor muy bajo de esta métrica.

Podemos observar que el modelo entrenado con el *training set* de sobremuestreo tiene el ROC AUC más elevado. Siguiendo el criterio explicado anteriormente, este será el modelo más eficiente.

4.1.4.2. Ajuste de hiperparámetros

Los hiperparámetros se utilizan para parametrizar el propio proceso de aprendizaje del modelo. Son las herramientas utilizadas para describir la configuración del modelo. Estos influyen en la capacidad y características de aprendizaje del modelo.

Se ha modificado el hiperparámetro del solver ('solucionador') y se ha regularizado la regresión logística.

- **Solver.**
  Los solvers son los algoritmos que hallan el valor mínimo de la función de costes. Por ejemplo, el solver newton-cg corresponde al "Newton conjugate gradient algorithm".

  Se realiza la validación con los cuatro solvers que presenta la librería scikit, los resultados se muestran en el cuadro 4.3.

  Se puede apreciar que según el resultado ROC AUC, los cuatro solvers ofrecen resultados parecidos, por lo que la eficiencia no depende mucho de dicho hiperparámetro. Pese a ello, el solver si que influye en el tiempo de ejecución. Los modelos que utilizan los solvers Newton-cg y Lbfgs tardan unas seis veces menos en ajustar el modelo que las otras dos.

	Newton-cg	**Lbfgs**	**Sag**	**Saga**
Tiempo	8,78125	6,71875	37,4375	38,7968
Exactitud	0,99943	0,99943	0,99948	0,99948
Precisión	0,75	0,75	0,93103	0,93103
Exhaustividad	0,9	0,9	0,75	0,75
F1 score	0,81818	0,81818	0,830769	0,830769
ROC AUC	0,87493	0,87493	0,87495	0,87495

Cuadro 4.3: Resultados según solver

De los cuatro modelos que se acaban de analizar ninguno supera el ROC AUC del modelo basado en overtraining, por lo que descartamos estos modelos.

- **Regularización.**

  La regularización es una técnica que sirve para evitar que nuestro modelo esté sobre-ajustado debido a los altos pesos asignados a ciertos parámetros que acompañan variables de alto grado. Se puede pensar la regularización como una penalización a la complejidad.

  La diferencia entre l1 y l2 de la función sklearn.preprocessing.normalize, trata en la penalización que le damos a los parámetros. L2 añade un término a la función de costes correspondiente al sumatorio de los parámetros θ al cuadrado. Esto penaliza altos grados de dichos parámetros, evitando la alta complejidad. L1 hace referencia a añadir a la función de costes el sumatorio de los parámetros θ directamente.

  Resultado de ello, la regularización a través de L2 reparte la penalización a todo los parámetros θ, mientras que L1 contará con parámetros nulos y otros que pueden ser elevados.

  Aplicamos ambas regularizaciones a la regresión logística y obtenemos las tablas de los cuadros 4.4 y 4.5 .

  Se puede apreciar que la regularización de los parámetros no tiene gran efecto en la mejora de la métrica ROC AUC. Las diferencias entre cada modelo son mínimas y ninguno supera el ROC AUC del modelo basado en overtraining, por lo que descartamos estos modelos.

L2 Penalty	**C=0,01**	**C=0,1**	**C=3**	**C=10**
Exactitud	0,99944	0,99948	0,99944	0,99944
Precisión	0,9	0,93103	0,9	0,89655
Exhaustividad	0,75	0,75	0,75	0,72222
F1 score	0,81818	0,83077	0,81818	0,8
ROC AUC	0,87493	0,87495	0,87493	0,86104

Cuadro 4.4: Regularización de parámetros mediante l2

L1 Penalty	**C=0,1**	**C=1**
Exactitud	0,99944	0,99944
Precisión	0,9	0,9
Exhaustividad	0,75	0,75
F1 score	0,81818	0,81818
ROC AUC	0,87493	0,87493

CUADRO 4.5: Regularización de parámetros mediante l1

4.1.4.3. Uso de la librería Tensorflow

Tensorflow es una de las librerías más populares en *Python* para el desarrollo de modelos de *Machine learning* y Deep Learning. Fue concebida por Google con el fin de minimizar el gap existente entre las aplicaciones de investigación y los productos software listos pare ser comercializados. A diferencia de otras librerías en el campo del cálculo numérico, como por ejemplo Numpy, Tensorflow fue especialmente ideado para simplicar la puesta a punto de modelos predictivos que impliquen una serie de operaciones secuenciales, además de facilitar el cálculo de ciertas cantidades (gradientes) útiles para llevar a cabo ciertas tareas de optimización. Existen otras librerías parecidas, como Theano, Pytorch, CNTK o Caffe pero sin lugar a dudas, Tensorflow es la más utilizada de todas ellas.

El principio de funcionamiento de Tensorflow se basa en el concepto de gráfica computacional y en la manipulación exclusiva de un tipo de objeto denominado tensor. En cuanto a las gráficas computacionales, la idea es que la gran mayoría de las operaciones se pueden descomponer en operaciones elementales (sumas, restas, productos, divisiones y funciones elementales). Como los modelos estadísticos de *Machine learning* son una secuencia de operaciones que a partir de los predictores, devuelven la predicción o estimación requerida, los modelos se pueden expresar con la API de Tensorflow en forma de gráfica computacional. Por otra parte, el tensor es un objeto muy adecuado para la minería de datos moderna ya que en ocasiones, el input del problema a resolver tiene más de una dimensión. Es el caso, por ejemplo, de un campo de la AI denominado Computer Vision, en el que input es un tensor de orden 3 (plano frontal de la imagen con 3 canales RGB).

Sin embargo, uno podría plantearse porque esta librería es tan adecuada con respecto a otras más clásicas que también pueden lidiar con cálculo vectorial. Por ejemplo, en teoría, sería también posible implementar modelos de *Machine learning* con Numpy. De hecho, probablemente Numpy tenga una comunidad de usuarios más grande. Con lo que, porque resulta más conveniente implementar modelos con Tensorflow? Existen varias razones:

- **Cálculo automático de gradientes.**
  Al definir la gráfica computacional, también conocido como el "forward propagation step" en la comunidad de Deep Learning, Tensorflow podría calcular automatícamente cualquier gradiente entre dos distintos nodos de la gráfica, mediante el uso de un algoritmo recursivo denominado "backpropagation". En consecuencia, implementar un modelo estadístico con Numpy resulta más engorroso puesto que hay que derivar explícitamente los gradientes y en ocasiones, pueden resultar bastante complicados.

- **Posibilidad de aprovechar GPUs.**
  Gran parte de los cálculos que intervienen en los modelos estadísticos son paralelizables. Por esta razón, la computación con tarjetas gráficas en vez de con procesadores (CPUs) suele ser ventajosa.

- **Una API más orientada a la minería de datos.**
  A diferencia de Numpy, una gran parte de las funciones contenidas en Tensorflow son realmente útiles para procesar información. En menos líneas de código, el programador es capaz de avanzar más rápidamente en el procesamiento de los datos.

Lógicamente, no todo podía ser positivo y uno de los inconvenientes que tiene Tensorflow es la no trivial curva de aprendizaje inicial, especialmente para usuarios de *Python* ya que la sintaxis presenta matices de C++ que pueden resultar poco familiares.

Una vez presentada la librería, se adjuntan los resultados obtenidos para regresión logística:

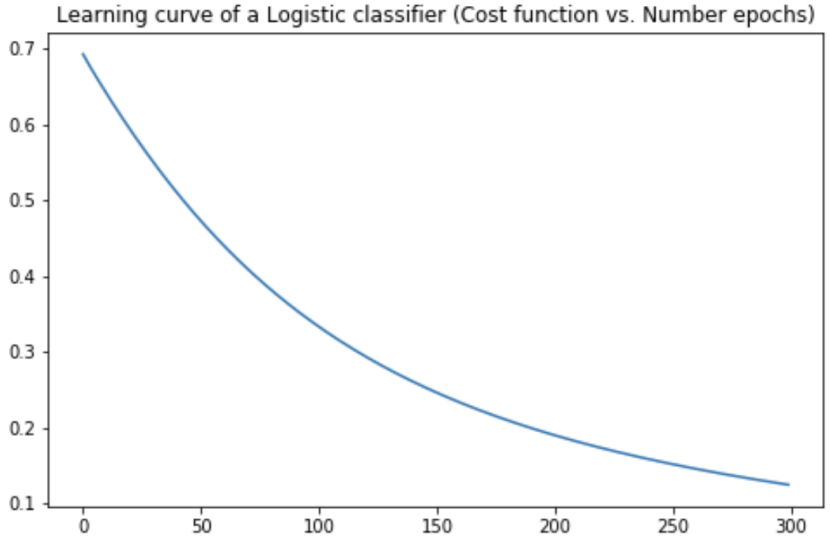


FIGURA 4.5: "Learning curve"de una regresión logística

Este resultado es clásico en el análisis de algoritmos de *Machine learning*. Se puede apreciar como inicialmente, la función de costes devuelva un valor elevado, lo que significa que los parámetros o "weights" provenientes de la inicialización no son los adecuados para reproducir las observaciones del *training dataset*. A medida que el optimizador local va proponiendo nuevos parámetros, el coste (output de la función de costes) decrece con el número de epochs. Una epoch es un término empleado para referirse a una pasada de todo el *training set*. La curva parece presentar una asíntota horizontal a grandes epochs, lo que indica que se alcanza una cota máxima en cuanto a capacidad del algoritmo a reproducir el *training dataset*.

Este resultado también demuestra el interés de utilizar la librería Tensorflow frente a otras como Scikit-learn. Da una mayor flexibilidad a la hora de poder calcular in-situ cualquier cantidad de interés (en este caso, la función de costes). Mientras que en Scikit-learn, por ejemplo, el usuario está limitado a los outputs que tengan sus funciones.

Por otra parte, es importante realizar una comparativa en términos de rendimiento final del algoritmo y tiempo de ejecución. Se adjunta la comparativa en la tabla 4.1.

Librería de ML	Precisión	Tiempo de ejecución
Scikit-learn (lbfgs)	0,99943	1,337s
Tensorflow	0,99939	$\sim 7s$

Cuadro 4.6: Comparativa entre Scikit-learn y Tensorflow

Se puede observar como la métrica de precisión proporciona resultados similares para Regresión Logística sobre el *training dataset* de fraude bancario y además muy elevados. Esto quiere decir que el algoritmo ha sido capaz de .ºverfit.el *training dataset*. Además, el tiempo de ejecución de Scikit-learn es 6 veces más rápido que con Tensorflow. La explicación puede encontrarse en el hecho de que los mejores resultados en tiempo de ejecución para Tensorflow se deben esperar cuando se utilizan GPUs y no CPUs, como ha sido el caso al ejecutar el experimento localmente en un portátil.

4.1.5. Modelo escogido

Con un ROC AUC de 0,96273, que le otorga una puntuación A como clasificador, se escoge el modelo entrenado mediante *oversampling*.

La matriz de confusión que presenta el modelo, para el conjunto de validación, es el siguiente:

Como última fase, aplicamos el modelo a nuestro *test set* para evaluar nuestro clasificador. La figura 4.7 muestra los resultados de la predicción.

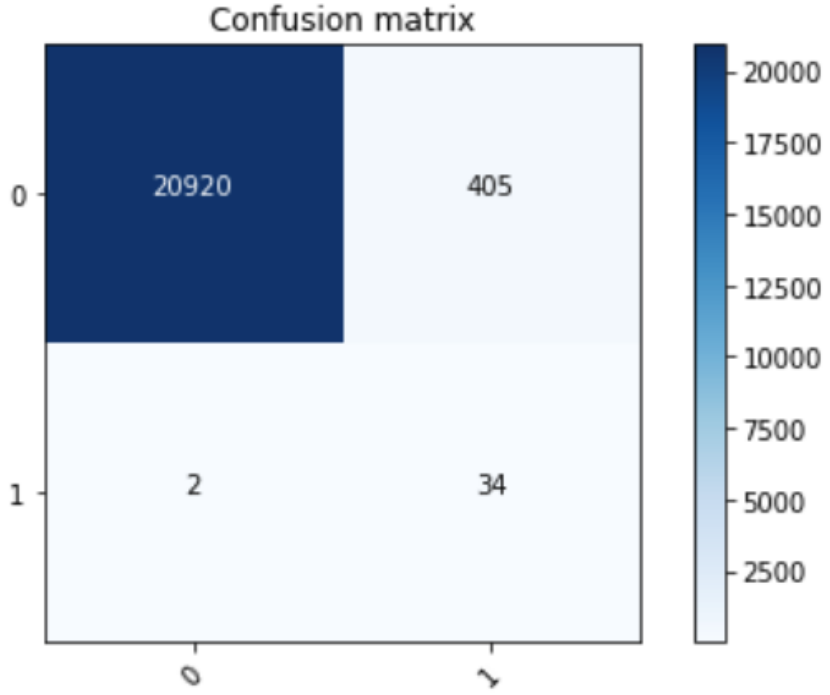


FIGURA 4.6: Matriz de confusión del modelo escogido para la etapa de validación

Se obtienen las métricas del cuadro

	Modelo RL con *oversampling*
Exactitud	0,98
Precisión	0,062222
Exhaustividad	0,84848
F1 score	0,11594
ROC AUC	0,91435

CUADRO 4.7: Métricas para el modelo escogido para el *test set*

Afortunadamente, el modelo escogido con un ROC AUC de 0,91435 se trata de un clasificador de tipo A o excelente.

4.2. Clasificación mediante *Support Vector Machines*

Para la clasificación mediante SVM, se han importado las mismas librerías, se ha realizado el mismo preproceso de datos, las métricas valoradas son las mismas puesto que el *dataset* está desequilibrado y la división en train, validation y *test set* es la misma.

La única diferencia está en que no se ha considerado equilibrar el *training set*. Se han realizados todos los entrenamientos con el *training set* desequilibrado. La mayor parte del trabajo se ha concentrado en la elección de los hiperparámetros.

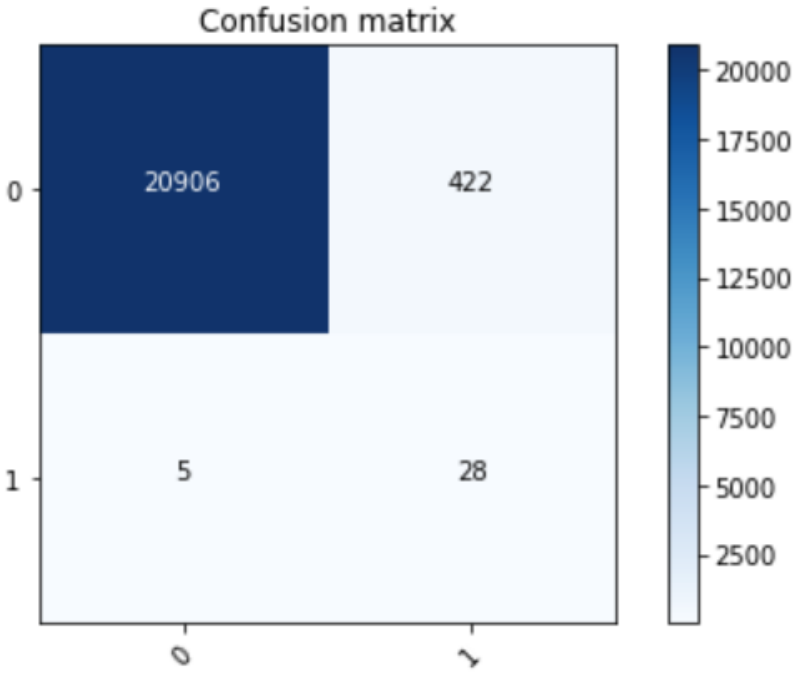


FIGURA 4.7: Matriz de confusión del modelo escogido para la etapa de prueba

4.2.1. Modelos estadísticos

4.2.1.1. Ajuste de hiperparámetros

Se han modificado los siguientes hiperparámetro: el tipo de kernel, dentro de los SVM con kernel de tipo polinómico se ha modificado el grado del polinomio, se han modificado los pesos específicos de las clases de un modelo con kernel rbf y por último se ha aplicado la función Grid Search para hallar los hiperparámetros óptimos a los modelos.

- **Tipos de kernel.**
  Los kernels permiten separar conjuntos de datos que en un espacio no son linealmente separables, a uno en que sí lo son. Se aplican los tres kernels que presenta la librería de Scikit learn (poly, rbf y sigmoide) para ver cual de ellos se ajusta mejor al *dataset*.

  Es importante destacar que los entrenamientos se han realizado con una muestra considerablemente menor de la que se cuenta en el *training set*, debido a los largos tiempos de ejecución. Para 100000 datos, el modelo lineal ha necesitado más de un minuto para procesarlo.

  Frente a los entrenamientos que superan los cinco minutos, se ha decidido utilizar menos datos. Los resultados se muestran en el cuadro 4.8.

Kernel	**Lineal**	**Polinómico**	**RBF**	**Sigmoide**
Tiempo	71,23	4,08	29,72	1,78
Exactitud	0,99949	0,99934	0,99925	0,99761
Precisión	0,8378	0,8666	0,9166	0,258064
Exhaustividad	0,86111	0,7222	0,61111	0,22222
F1 score	0,84932	0,78787	0,7333	0,23887
ROC AUC	0,930415	0,861	0,80551	0,61057

CUADRO 4.8: Resultados según kernel

El modelo lineal responde estupendamente, con un ROC AUC superior a 0,9. Por otra parte el tiempo de ejecución ha sido el más elevado de todos, con diferencia. Le sigue el modelo ajustado al kernel polinómico con un ROC AUC de 0,86 y un tiempo de ejecución 17 veces más rápido. El modelo sigmoide no presenta unas buenas métricas y el modelo RBD (Radial Basis Function) no destaca frente al modelo linear o con aplicación de kernel polinómico.

Por lo tanto, de momento se mantendrá el modelo lineal como modelo favorito.

- **Grado del kernel polinómico.**
  A la hora de utilizar el kernel polinómico, como en el ejemplo anterior, se puede modificar el grado del polinomio. Por defecto se encuentra en grado 3. Se ha realizado el análisis con 4 grados diferentes, los resultados se presentan en el cuadro 4.9.

Grado polinomio	**1**	**2**	**5**	**10**
Exactitud	0,999532	0,99953	0,99929	0,99887
Precisión	0,84211	1	0,8	0,63636
Exhaustividad	0,888	0,7222	0,777	0,7777
F1 score	0,86486	0,8387	0,78873	0,7
ROC AUC	0,944303	0,86111	0,888724	0,88851

CUADRO 4.9: Resultados según grado kernel polinómico

Se puede observar que los cuatro algoritmos entrenados con un kernel polinómico, de grados diferentes, son de categoría B o superior.

El modelo entrenado con kernel polinómico de grado 1 cuenta con un ROC AUC de 0,9443, el mejor de los modelos planteados hasta ahora. Este pasa a ser nuestro nuevo modelo favorito.

- **Pesos asignados a las clases.**
  La función svm.SVC de sklearn presenta un hiperparámetro que permite regular la penalización que se le atribuye a una clasificación errónea de una clase. Este parámetro corresponde a Class Weights o pesos de las clases.

Poniendo el caso en el que se asigna todo el peso (importancia) a la clase positiva, se produce la siguiente clasificación (Figura 4.8). Se observa que dicha configuración no permite un solo error en la clasificación de clases positivas (FN=0), en contrapartida, no se predicen correctamente las transacciones no fraudulentas.

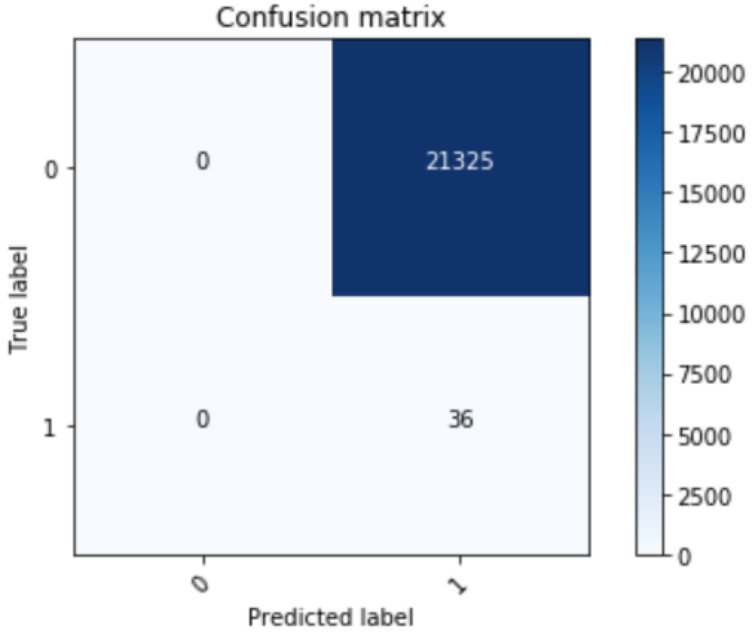


FIGURA 4.8: Máxima penalización a la clase negativa

Por otra parte, si se le asigna todo el peso a la clase negativa, se produce la siguiente clasificación (Figura 4.9). Este escenario es equivalente al modelo que, para maximizar la exactitud, clasifica todas las transacciones como no fraudulentas. Provocando que todos los fraudes pasen desapercibidos.

Conociendo estas dos situaciones límite, se ha creado una función que recorre todos los pesos incluidos entre 0 y 1, con un paso de 0,01, en ordenes opuestos (creciente, decreciente). Para cada pareja de pesos (peso asociado a la clase negativa y positiva) se entrena el modelo y se recuperan los FN y FP una vez hecha la predicción. [Anexo C]. En la figura 4.10, se aprecia el resultado de este análisis.

Esta gráfica puede recordar a la curva ROC comentada en el apartado métricas. Esto se debe a que las ordenadas correspondes a FP y las coordenadas a FN. En el espacio ROC se representan los FP y TP. Este análisis se ha realizado con estas dos variables ya que son las que se deben minimizar. Se aprecia que el aumento de fallos de una clase recae en la disminución de fallos en la clase complementaria. Mediante esta gráfica se puede determinar cual es el equilibrio que se desea mantener entre ambas clases.

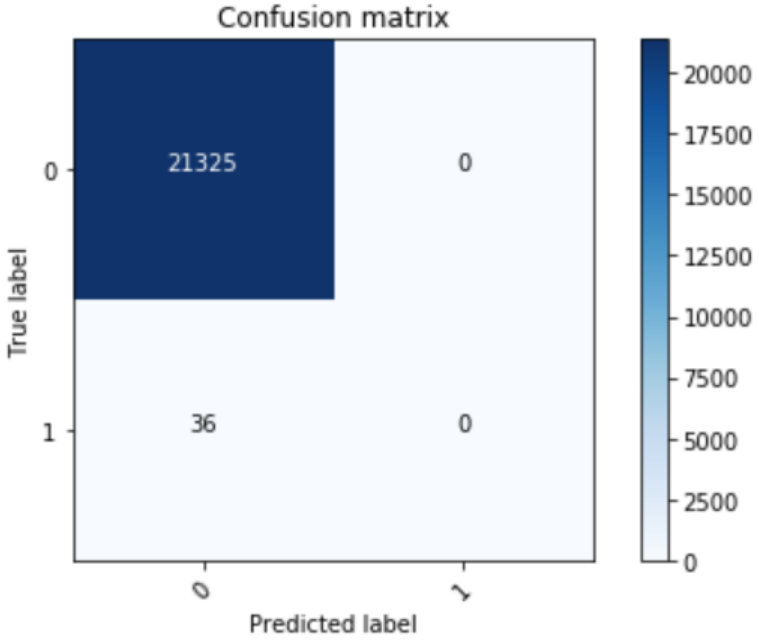


FIGURA 4.9: Máxima penalización a la clase positiva

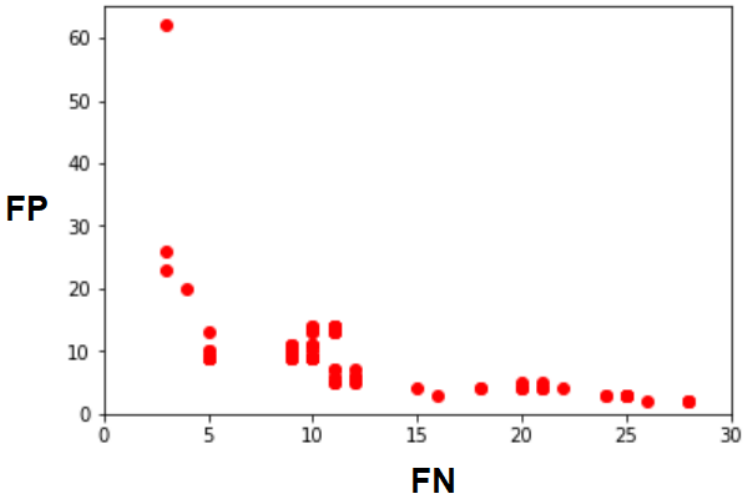


FIGURA 4.10: Efecto de la variación de los pesos sobre las predicciones

- **Gridsearch.**

  Por último se ha utilizado la función GridSearch que permite hallar los hiperparámetros C y γ óptimos para un modelo que emplea un kernel rbf.

  En primer lugar se hallarán los parámetros a través de Gridsearch y luego se comparará la eficiencia del modelo ajustando los parámetros, frente al anterior modelo sin ajustar.

  Destacar el hecho de que se ha tenido que utilizar una parte del *training set*, para evitar tiempos de ejecución muy elevados, al usar la función Gridsearch.

Mediante la función, se han hallado que los parámetros óptimos son: $\{C = 1; \gamma = 0,01\}$. Entrenando el modelo de SVM, con el kernel rbf optimizado se obtienen los resultados de la figura .

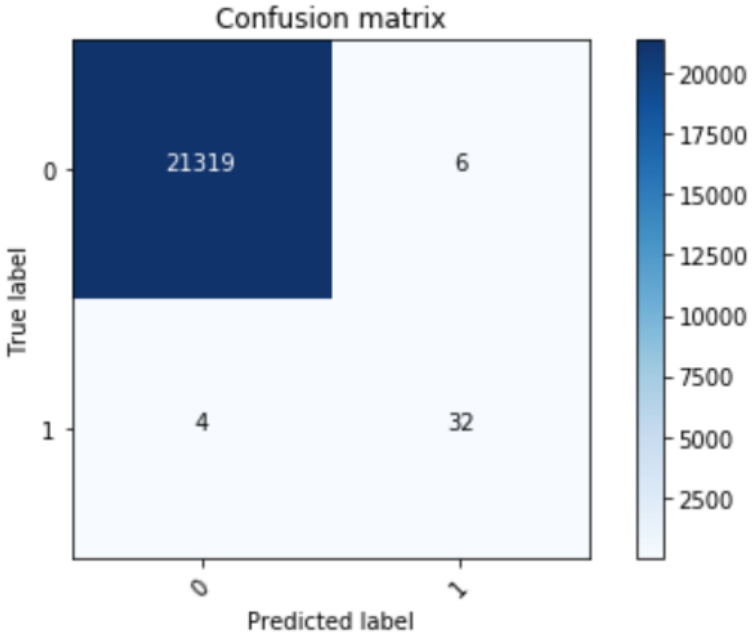


FIGURA 4.11: Matriz de confusión para el modelo de kernel rbf optimizado

Dicho modelo presenta un ROC AUC de 0,944, y unas métricas iguales al del kernel polinómico de grado 1. Estos dos modelos han resultado ser los más eficientes frente al *validation set*.

Para realizar la prueba, se opta por escoger el modelo con kernel rbf optimizado.

4.2.2. Modelo escogido

Con un ROC AUC de 0.944, que le otorga una puntuación A como clasificador, se escoge el modelo entrenado con kernel rbf optimizado.

Como última fase, aplicamos el modelo a nuestro *test set* para evaluar nuestro clasificador. La figura muestra los resultados de la predicción.

Se obtienen las métricas del cuadro 4.10.

Desgraciadamente el modelo clasifica peor en el conjunto de prueba, el modelo escogido pasa a tener un ROC AUC de 0.78769 que describe el clasificador como tipo B o bueno.

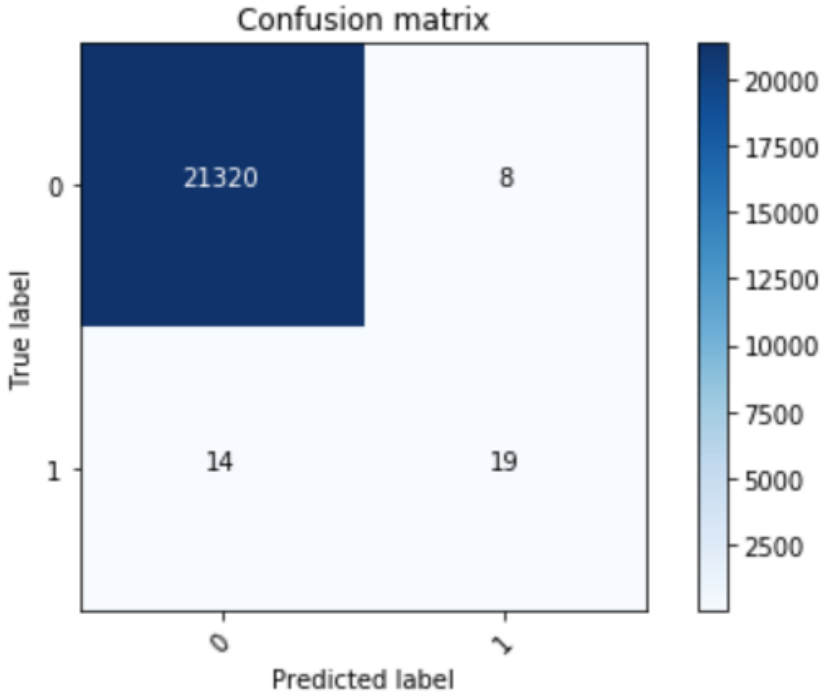


FIGURA 4.12: Matriz de confusión del modelo escogido para la etapa de prueba

	Modelo SVM con kernel RBF optimizado
Exactitud	0,9987
Precisión	0,7037
Exhaustividad	0,57575
F1 score	0,63334
ROC AUC	0,78769

CUADRO 4.10: Métricas para el modelo escogido para el *test set*

4.3. Comparación de resultados

Se han aplicado dos algoritmos para la clasificación de transacciones en calidad de fraudulentas o no fraudulentas. A través del análisis del *dataset*, el modelo más eficiente de regresión logística se ha conseguido siendo entrenado mediante un *training set* de sobremuestreo. Por otra parte, mediante el ajuste de hiperparámetros, se ha hallado la configuración óptima para un modelo basado en kernel RBF. En la figura 4. se muestra la comparativa de la eficiencia de los modelos, aplicados al *dataset* de prueba.

Siguiendo el criterio establecido en todo el proyecto, compararemos ambos modelos comparando el ROC AUC. Se puede apreciar que el modelo de regresión logística supera con creces al de SVM. Por lo tanto, el modelo que se debe emplear para la transacción de fraude es un modelo de regresión logística entrenado por sobremuestreo.

	Modelo RL con *oversampling*	Modelo SVM con kernel RBF optimizado
Exactitud	0,98	0,9987
Precisión	0,062222	0,7037
Exhaustividad	0,84848	0,57575
F1 score	0,11594	0,63334
ROC AUC	0,91435	0,78769

CUADRO 4.11: Métricas para los modelo escogidos

Por otra parte, se puede adoptar un criterio basado en razones económicas. En el siguiente apartado se detalla el escenario.

4.3.1. Criterio económico

Desde la prueba en el modelo de regresión logística se han obtenido 5 FN y 422 FP. Por otra parte, con el modelo de SVM se ha obtenido 14 FN y 8 FP.

Se puede realizar una valoración económica del valor de un FP y un FN.

Considerando que el salario medio de una persona que se dedica a la atención al cliente es de 1200UM, eso equivale a un precio por hora de 7,5UM.

Un falso negativo implica que ha sucedido una transacción fraudulenta y no se ha detectado. Teniendo en cuenta que el promedio de la cantidad defraudada, en el *dataset* con el que se ha trabajado, es de 122,21UM, se trata del gasto que supone a la empresa sufragar el reembolso a la persona afectada. Por otra parte, considerando que la empresa reciba en promedio una llamada de aproximadamente una hora para resolver problemas relacionados con el fraude, eso supone 7,5UM añadidos por fraude. ($Coste_{FN} = 129,71\ UM$)

Un falso positivo implica que a un cliente le han impedido realizar una transacción. En dicha situación, los clientes se estima que de promedio son atendidos durante 20 minutos para despejar las dudas. Estos 20 minutos representan un coste se 3UM. ($Coste_{FP} = 3\ UM$)

Si realizamos un balance del gasto que supondría la clasificación por parte de la regresión logística y de los SVM, se obtendría lo siguiente:

$$Coste_{RL} = 5 * Coste_{FN} + 422 * Coste_{FP} = 1914,55\ UM$$
$$\text{Coste}_{SVM} = 14 * Coste_{FN} + 8 * Coste_{FP} = 1839,94\ UM$$

Podemos observar que siguiendo los criterios que se han expuesto, a modo de intuición, resulta ser el modelo basado en SVM más adecuado, ya que presentaría un menor

coste para la empresa. Un modelo, que sería útil para realizar este análisis económico, corresponde al modelo en el que se ha ido variando el peso de las clases (SVM).

En resumen, es de suma importancia tener claro los objetivos de la optimización puesto que cada objetivo cuenta con su propio estudio. Los modelos pueden ser mejores o peores según la naturaleza de la problemática.

Capítulo 5

Conclusiones

En este proyecto se ha abordado todo el proceso de implementación, de un modelo estadístico, a un *dataset*. Se ha cubierto desde la parte teórica de los algoritmos, hasta la propia implementación en *Python*.

En primer lugar, se ha realizado un análisis del *dataset* con el que se contaba. Este se encontraba completamente desequilibrado, una de las clases constituía más de un 95 % de las muestras (transacciones no fraudulentas). Cabe destacar que el tratamiento previo de la información es fundamental para obtener buenos resultados.

Con el objetivo de equilibrar el *dataset*, se han aplicado las técnicas de *oversampling* y *undersampling*. Y hemos podido observar, que el modelo de regresión logística entrenado con un *dataset* de sobreajuste obtiene mejores resultados que con un *dataset* desequilibrado.

Por otra parte, se ha indagado en los hiperparámetros de los algoritmos y realizado diversas pruebas para analizar su efecto. Se ha llegado a la conclusión que a través de un ajuste adecuado de dichos parámetros, se pueden lograr mejores resultados (véase el modelo con kernel rbf).

Es importante destacar el tiempo de procesamiento de dichos algoritmos, según el método de aprendizaje, el ajuste de los modelos tardan más o menos. Con una muestra inferior a 100.000 datos, ciertos algoritmos han tardado tiempos superiores a los cinco minutos en ajustar el modelo. Si se escala dicha situación a un caso muchos más datos, se deberá tener en cuenta.

No hay que olvidarse de prestar atención a los fenómenos de *overfitting* y *underfitting* en los casos que puedan aparecer. Se deben evitar este tipo de extremos.

El potencial que tienen los algoritmos de aprendizaje supervisado es enorme. La industria 4.0 se beneficia de ellos y se están desarrollando más aplicaciones. Estos algoritmos forman parte del motor de cambio en la Industria 4.0. Cómo aplicaciones se puede mencionar la detección de fatiga en los conductores, la autenticación segura en las apps móviles, controlar plagas endémicas o la conducción autónoma.

FIGURA 5.1: Proyecto de conducción autónoma por *drive.ai*

Debido a lo positiva que ha sido la experiencia de adentrarse en el mundo del *Machine Learning*, tras este proyecto se seguirá profundizando con el manejo de librerías más sofisticadas, como sería *Tensorflow*.

Anexo A

Hiperplanos

En geometría, un hiperplano es un subespacio que tiene una dimensión menos que la del espacio en el que se encuentra. Existen distintos tipos de hiperplanos, entre otros: los afines, los de vector y los proyectivos. Los hiperplanos solución de los SVM pertenecen a la categoría de afines, estos últimos serán detallados a continuación.

En coordenadas cartesianas, un hiperplano afín puede definirse con una única ecuación lineal, de la siguiente manera (donde al menos un parámetro a es no nulo):

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b.$$

En un espacio afín, donde las coordenadas son números reales, el hiperplano separa el espacio en dos partes. Estas dos partes se definen con las siguientes inecuaciones:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n < b$$
$$a_1x_1 + a_2x_2 + \cdots + a_nx_n > b.$$

En un espacio tridimensional los hiperplanos corresponderían a planos (ver figura A.3), equivalentes a subespacios bidimensionales. En el caso de un espacio bidimensional los hiperplanos son rectas (ver figura A.2). Por último, en un espacio unidimensional, el hiperplano corresponde a un punto (ver figura A.1). En las figuras A, el término H hace referencia al hiperplano asociado.

FIGURA A.1: Espacio unidimensional

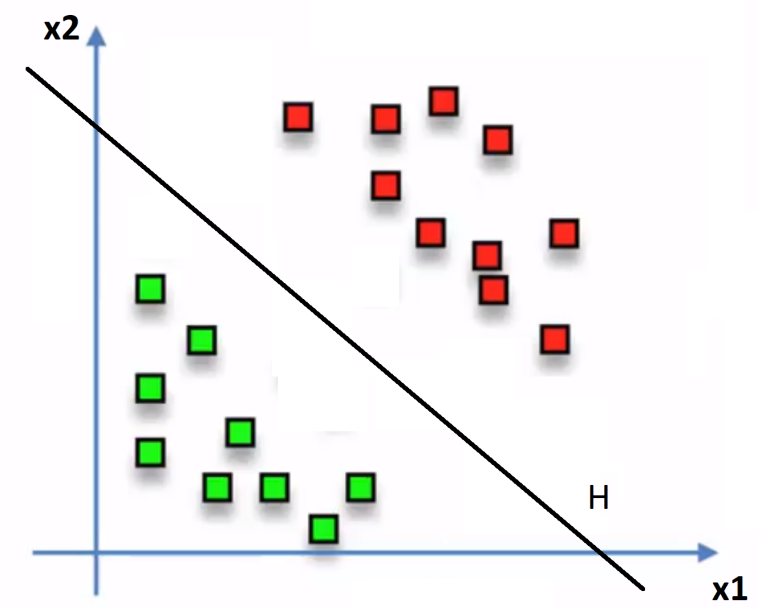


FIGURA A.2: Espacio bidimensional

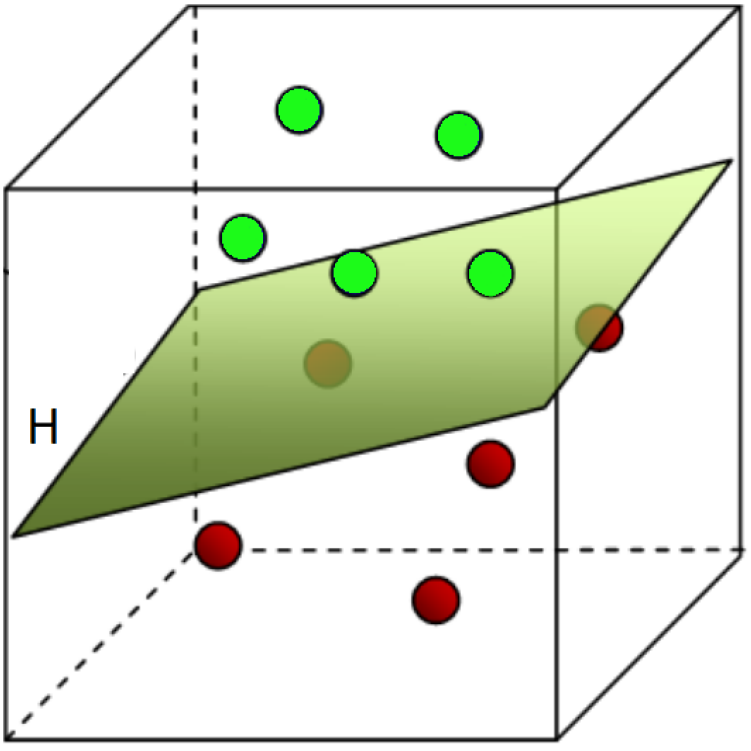


FIGURA A.3: Espacio tridimensional

Anexo B

Underfitting y Overfitting

En estadística, el concepto de *overfitting* o sobreajuste es la producción de un modelo que corresponde de manera muy cercana o exacta a un conjunto especifico de datos (Figura B.1, tercer gráfico). Este modelo tan ajustado a los datos puede fallar a la hora de realizar predicciones con nuevas muestras, se dice que dicho modelo contiene más parámetros de los que pueden ser justificados por el conjunto de datos. Básicamente, los datos que podrían formar parte del ruido son integrados en el modelo. Un modelo sobreajustado presenta una alta varianza pero un reducido sesgo estadístico.

La varianza es la cuantificación de la tendencia de un modelo a cambiar en función del conjunto de datos de entrenamiento, debemos evitar que sea muy elevada. Dicho parámetro describe la complejidad del modelo. El sesgo estadístico es un error que se detecta en los resultados de un estudio, se trata de las clasificaciones erróneas en el caso de un clasificador. Dicho parámetro cuantifica el error presente en un modelo.

El concepto de *underfitting* o subajuste ocurre cuando un modelo no es capaz de capturar la estructura de los datos (Figura B.1, primer gráfico). En contraposición al *overfitting*, un modelo que presenta *underfitting* contiene menos parámetros de los que pueden ser justificados por los datos. Este fenómeno puede suceder cuando intentamos ajustar un conjunto de datos no lineal con un modelo lineal. Dicho modelo tendrá una eficiencia de clasificación no deseada. Un modelo subajustado presenta una baja varianza pero un elevado sesgo estadístico.

A la hora de generar un modelo, se tiende a buscar un equilibrio entre la varianza y los sesgos estadísticos. Este equilibrio se puede observar en el segundo gráfico de la Figura B.1.

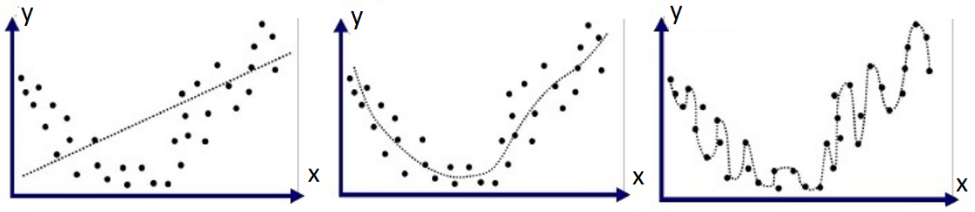


FIGURA B.1: Diferentes modelos para un mismo conjunto de datos

Los fenómenos del *overfitting* y *underfitting* pueden suceder en machine learning, estos últimos pueden recibir el nombre de *overtraining* (sobreentrenamiento) y *undertraining* (subentrenamiento).

En la siguiente figura, aparecen tres gráficos con el mismo conjunto de datos, los datos se separan en dos clases y se aprecian tres modelos de clasificación distintos. El primer gráfico presenta *overfitting*, el segundo se ajusta bastante bien al conjunto de datos y el tercero presenta *underfitting*. Se aprecia claramente que el modelo sobreajustado es más complejo pero tiene menos errores respecto al subajustado.

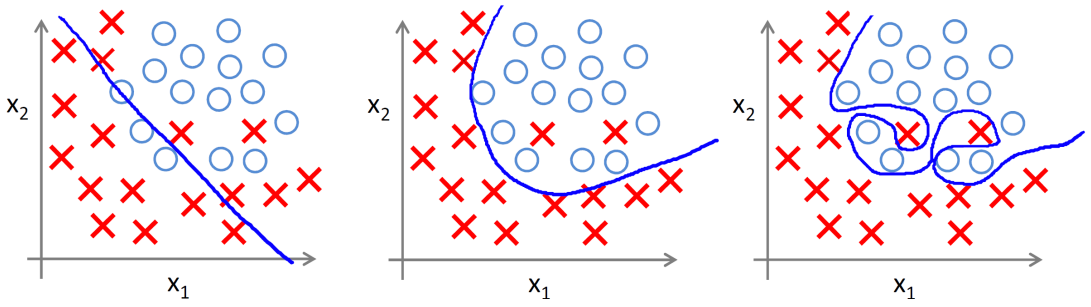


FIGURA B.2: Diferentes modelos de clasificación para un mismo conjunto de datos

Anexo C

Código Regresión Logística

En el siguiente anexo se detalla el código empleado para clasificar el conjunto de datos de estudio, mediante el algoritmo de regresión logística.

En el capítulo 5 se analizan los diferentes *outputs* que produce el siguiente código.

0.1 Import libraries

```
In [1]: import numpy as np # biblioteca de funciones matemáticas de alto nivel para operar
                           #con esos vectores o matrices.
        import pandas as pd # ofrece estructuras de datos y operaciones para manipular tablas nu
        from sklearn.preprocessing import StandardScaler
        from collections import Counter
        import itertools

        #General tools
        import time
        import sys

        #Machine Learning Models
        from sklearn.linear_model import LogisticRegression

        #Metrics
        from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_
        from sklearn.metrics import accuracy_score, recall_score, precision_score

        #Plotting
        import matplotlib #biblioteca para la generación de gráficos a partir de datos contenido
        import matplotlib.pyplot as plt #plt.plot(x, y) implica coord y ord grafico
        import seaborn as sns #libreria de visualización

        %matplotlib inline
        #guardar los graficos
```

```
In [2]: from IPython.core.interactiveshell import InteractiveShell #que salgan todos los prints
        InteractiveShell.ast_node_interactivity = "all"
```

0.2 Data loading

```
In [3]: data = pd.read_csv("creditcard.csv") #subir los datos
        data.head() #las primeras filas de una tabla
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	\
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	

	V8	V9	...	V21	V22	V23	V24	\
0	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	
1	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	
2	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	
3	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	
4	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	

```
        V25       V26       V27       V28  Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1  0.167170  0.125895 -0.008983  0.014724    2.69      0
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3  0.647376 -0.221929  0.062723  0.061458  123.50      0
4 -0.206010  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

0.3 Data visualization/analysis

```
In [4]: data.columns
```

```
Out[4]: Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
               'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
               'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
               'Class'],
              dtype='object')
```

```
In [5]: len(data) #dateframe es data, una tabla, este es el numero de filas contando la cabecera
```

```
Out[5]: 284807
```

```
In [6]: count_class=data.Class.value_counts() #cuenta los distintos valores de la columna, panda
        #se ve que claramente esta imbalanced
        print(count_class)
```

```
0    284315
1       492
Name: Class, dtype: int64
```

```
In [7]: #grafico de barras para ver la distribución de clases
        plt.figure(1)
        plt.figure(figsize=(5,8))

        count_class.plot(kind='bar')
        plt.ylabel('Clase Binaria')
        plt.title('Numero de ejemplos por clase en el dataset')
```

```
Out[7]: <matplotlib.figure.Figure at 0x28b21dbd898>
```

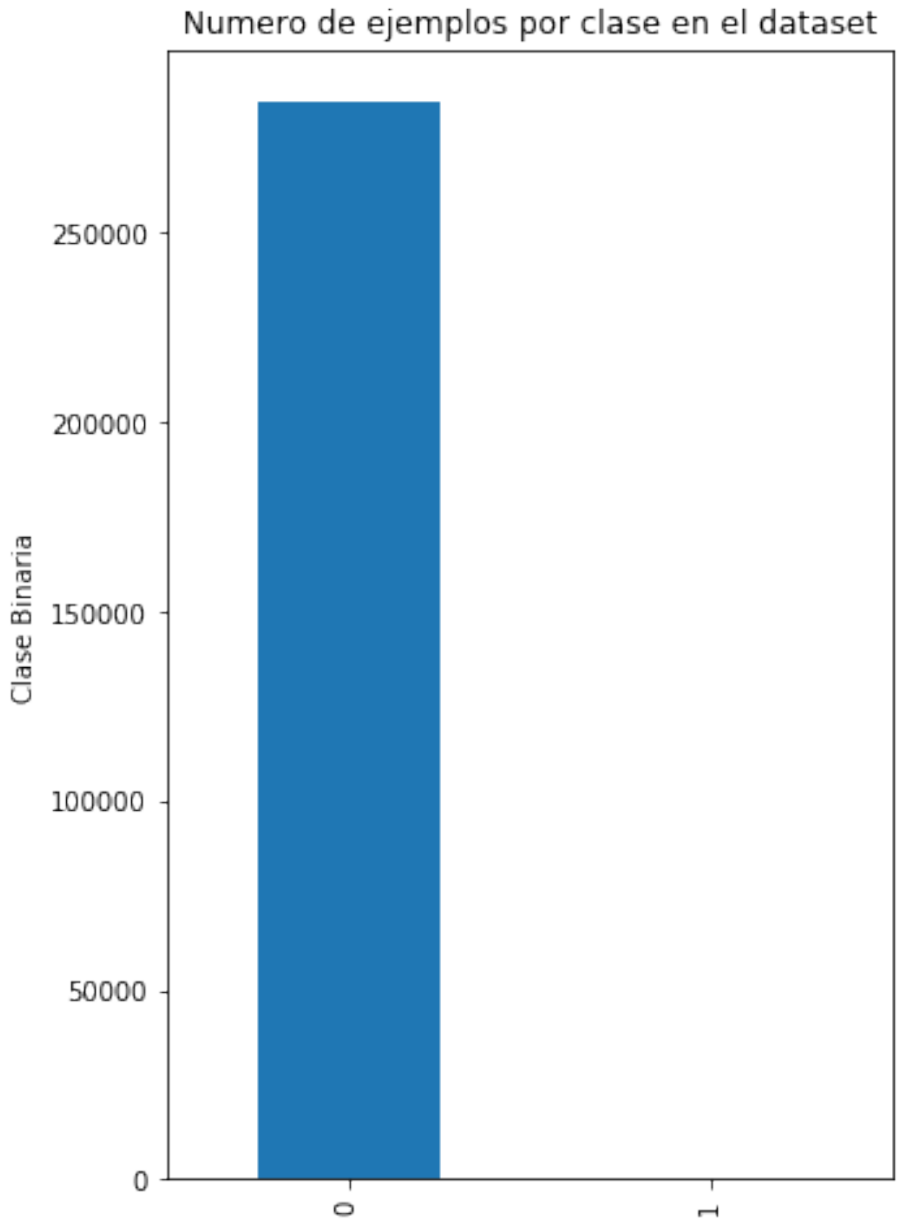
```
Out[7]: <matplotlib.figure.Figure at 0x28b21d93eb8>
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x28b21dde1d0>
```

```
Out[7]: Text(0,0.5,'Clase Binaria')
```

```
Out[7]: Text(0.5,1,'Numero de ejemplos por clase en el dataset')
```

```
<matplotlib.figure.Figure at 0x28b21dbd898>
```




```
In [8]: # Compute some basic statistics from the two non-normalized features

        data[['Time', 'Amount']].describe() #te da la suma, media, min max etc
```

Out[8]:

	Time	Amount
count	284807.000000	284807.000000

```
mean     94813.859575      88.349619
std      47488.145955     250.120109
min          0.000000       0.000000
25%      54201.500000       5.600000
50%      84692.000000      22.000000
75%     139320.500000      77.165000
max     172792.000000   25691.160000
```

In [9]:
```
sc = StandardScaler() #estandariza las variables eliminando la media y escalando la vari
data['s_Amount']=sc.fit_transform(data['Amount'].values.reshape(-1,1)) #s_amount esta tr

data[['Amount', 's_Amount']].head()
```

Out[9]:

	Amount	s_Amount
0	149.62	0.244964
1	2.69	-0.342475
2	378.66	1.160686
3	123.50	0.140534
4	69.99	-0.073403

In [10]:
```
data = data.drop(['Time', 'Amount'], axis=1) #borras las columnas
data.columns
```

Out[10]:
```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Class', 's_Amount'],
      dtype='object')
```

In [11]:
```
class_names=['0','1']
#definición de la matriz de confusión
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
```

```
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.show()

    def show_data(cm, print_res = 0):
        tp = cm[1,1]
        fn = cm[1,0]
        fp = cm[0,1]
        tn = cm[0,0]
        if print_res == 1:
            print(tp)
            print(tn)
            print(fp)
            print(fn)
            print('Accuracy =      {:.3f}'.format((tn+tp)/(tp+fp+tn+fn)))
            print('Precision =     {:.3f}'.format(tp/(tp+fp)))
            print('Recall (TPR) =  {:.3f}'.format(tp/(tp+fn)))
            print('Fallout (FPR) = {:.3e}'.format(fp/(fp+tn)))
        return tp/(tp+fp), tp/(tp+fn), fp/(fp+tn)
```

0.4 Train/val/test set split

In [12]:
```
#separamos en variables x,y
x = data.loc[:, data.columns != 'Class'] #pedir datos de la tabla, : implica todas las
y = data.loc[:, data.columns == 'Class']
#separamos en inputs y outputs
```

In [13]:
```
#Split the data into train and test data

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.15, random_state =

x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.5, random_s
```

In [14]:
```
print(len(x))
print(len(x_train))
print(len(x_test))
print(len(x_val))
```

```
284807
242085
21361
```

21361

0.5 Data preprocessing

0.5.1 Undersampling of the dataset

In [15]:
```
#1. Find the number of the minority class
len(y_train)
number_fraud = sum(y_train['Class'])
number_non_fraud = len(y_train)-number_fraud
print(number_fraud)
print(number_non_fraud)
```

Out[15]: 242085

423
241662

In [16]:
```
#2. Find the indices of the majority class
index_non_fraud = y_train[y_train['Class']==0].index  #LISTA CON LAS POSICIONES
#.3 Find the indices of the minority class
index_fraud = y_train[y_train['Class']==1].index
print(len(index_non_fraud)+len(index_fraud))
```

242085

In [17]:
```
#4. Randomly sample the majority indices with respect to the number of minority classes
random_indices = np.random.choice(index_non_fraud, number_fraud,replace='False') #prime
len(random_indices)
```

Out[17]: 423

In [18]:
```
#5. Concat the minority indices with the indices from step 4
under_sample_indices = np.concatenate([index_fraud,random_indices])
len(under_sample_indices)/2
```

Out[18]: 423.0

In [19]:
```
#Get the balanced dataframe - This is the final undersampled data
under_sample_df = data.iloc[under_sample_indices] #Purely integer-location based indexi
under_sample_df
```

Out[19]:

	V1	V2	V3	V4	V5	V6	\
14104	1.192396	1.338974	-0.678876	3.123672	0.643245	-1.184323	
43061	-15.020981	8.075240	-16.298091	5.664820	-11.918153	-4.246957	
74507	-7.427924	2.948209	-8.678550	5.185303	-4.761090	-0.957095	
276071	2.091900	-0.757459	-1.192258	-0.755458	-0.620324	-0.322077	

42473	-3.600544	4.519047	-6.340884	6.214767	-5.829558	-2.478095
184379	-1.141559	1.927650	-3.905356	-0.073943	-0.044858	-1.756999
234632	1.261324	2.726800	-5.435019	5.342759	1.447043	-1.442584
143335	-6.713407	3.921104	-9.746678	5.148263	-5.151563	-2.099389
10690	-12.224021	3.854150	-12.466766	9.648311	-2.726961	-4.445610
12108	-16.917468	9.669900	-23.736443	11.824990	-9.830548	-2.514829
30100	-3.218952	2.708535	-3.263042	1.361866	-1.645776	-1.852982
27749	-0.860827	3.131790	-5.052968	5.420941	-2.494141	-1.811287
281674	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695
76555	-7.901421	2.720472	-7.885936	6.348334	-5.480119	-0.333059
191359	1.177824	2.487103	-5.330608	5.324547	1.150243	-1.281843
108258	0.196707	1.189757	0.704882	2.891388	0.045555	1.245730
17407	-29.200329	16.155701	-30.013712	6.476731	-21.225810	-4.902997
152019	-3.705856	4.107873	-3.803656	1.710314	-3.582466	1.469729
191267	0.290155	0.049243	-0.740524	2.865463	1.395294	-0.535163
238222	-4.280584	1.421100	-3.908229	2.942946	-0.076205	-2.002526
6899	-2.661802	5.856393	-7.653616	6.379742	-0.060712	-3.131550
154587	-28.255053	21.467203	-26.871339	11.737436	-17.999630	6.065901
208651	0.630579	1.183631	-5.066283	2.179903	-0.703376	-0.103614
8842	-4.696795	2.693867	-4.475133	5.467685	-1.556758	-1.549420
149874	-1.662937	3.253892	-7.040485	2.266456	-4.177649	-0.746925
189701	-4.599447	2.762540	-4.656530	5.201403	-2.470388	-0.357618
6971	-3.499108	0.258555	-4.489558	4.853894	-6.974522	3.628382
6820	-2.169929	3.639654	-4.508498	2.730668	-2.122693	-2.341017
79525	-2.630598	5.125759	-6.092255	5.527393	1.605145	-2.319884
192687	1.522080	-0.519429	-2.581685	0.774741	0.206722	-1.431020
...	...	...	...	...	...	...
10337	-0.958604	0.001013	2.364540	1.241371	0.147725	0.885649
115396	-1.098505	0.141628	1.180778	-3.237091	1.096154	-1.350058
88347	-0.505975	0.822155	0.455688	0.127137	-0.502624	-0.864967
243657	2.266550	-1.399208	-0.557516	-1.437509	-1.598576	-0.919531
274660	-0.383510	0.211535	2.423758	1.448221	-0.569553	0.905035
214155	-1.062592	1.402230	-0.333878	-0.992051	0.251246	-0.723824
240014	2.140138	-1.177022	-0.789782	-0.904917	-0.993683	-0.300402
59100	1.170334	0.198286	1.178273	2.707085	-0.302594	0.892273
241384	2.075478	-0.023523	-1.276101	0.356568	-0.060071	-1.302219
206068	-0.240561	0.260887	0.096938	-1.395639	2.321155	4.166307
50249	1.135426	0.608113	0.205817	2.598191	-0.041364	-1.034539
248184	-0.342693	0.992944	-1.988845	0.113088	4.031550	3.156955
233242	-0.308850	0.686057	-2.710482	-1.035567	2.021087	3.351639
157333	-1.169640	0.270381	2.094015	-0.340563	0.578368	0.342654
202540	1.960762	-0.851750	-0.989180	-1.077912	-0.280340	0.257511
173688	1.973834	-0.547721	-0.532265	0.245136	-0.457054	0.187423
26359	-3.097874	-2.983912	1.413452	-1.021595	-0.486538	-0.009405
216148	-16.197900	-22.159993	-1.030403	3.278934	14.555466	-9.295025
94662	-4.193541	-2.914787	1.570264	1.677741	1.355149	-0.206745
172899	1.927917	-1.405052	-0.749556	-1.073929	-0.892127	0.214317
21933	-0.569781	-0.617292	1.448933	-2.500217	-0.610863	-0.593342

```
112627  -0.764109   1.319467   1.738212   0.973438  -0.418212 -0.314742
216440   2.062347  -0.076676  -1.492637   0.143447   0.236249 -0.776411
141899  -0.724379   0.000377   2.024790  -0.985825   0.142039  0.211408
212511  -0.721263   0.328141   0.400280   0.964958  -0.073716  0.782448
220937  -1.218992   0.843750   1.019641  -0.253716   1.349089 -0.544659
4071    -0.995358   1.505073   1.027121   0.141408  -0.207449 -0.358016
213129   1.969512  -0.846761  -1.122599  -0.395917  -0.409531 -0.183475
242789   1.741758  -0.403864  -2.070451   0.443218   0.226827 -0.888560
124375  -0.723049   0.804992   2.113530   0.406375   0.189162 -0.560182

                V7         V8         V9        V10     ...           V21  \
14104     0.397586  -0.253499   0.411135  -0.859862     ...     -0.377503
43061   -14.716668   9.435084  -6.795398 -15.124163     ...      2.525115
74507    -7.773380   0.717309  -3.682359  -8.403150     ...     -0.299847
276071   -1.082511   0.117200  -0.140927   0.249311     ...      0.288253
42473    -9.938412   2.830086  -5.659162 -11.298156     ...      2.263770
184379   -1.217416   0.364563  -2.770148  -3.216188     ...      0.102081
234632   -0.898702   0.123062  -2.748496  -3.202436     ...      0.209086
143335   -5.937767   3.578780  -4.684952  -8.537758     ...      0.954272
10690   -21.922811   0.320792  -4.433162 -11.201400     ...     -1.159830
12108   -17.290657   1.820408  -6.264903 -12.916636     ...     -2.336111
30100    -3.069958  -1.796876  -0.213356  -3.551984     ...      1.807877
27749    -5.479117   1.189472  -3.908206  -7.060746     ...      1.192694
281674    0.223050  -0.068384   0.577829  -0.888722     ...     -0.164350
76555    -8.682376   1.164431  -4.542447  -7.748480     ...      0.077739
191359   -1.171994   0.413778  -2.659840  -2.971695     ...      0.262325
108258   -1.198714  -2.421616  -1.232089   0.324239     ...     -1.328132
17407   -19.791248  19.168327  -3.617242  -7.870122     ...      1.809371
152019   -9.621560 -11.913105  -0.322297  -6.625692     ...     -5.498772
191267    0.142543  -0.222770  -1.463691   1.713538     ...      0.337349
238222   -2.874155  -0.856005   0.963674  -3.235439     ...     -0.140062
6899     -3.103570   1.778492  -3.831154  -7.191604     ...      0.734775
154587  -41.506796 -38.987263 -13.434066 -24.403185     ...    -21.453736
208651   -3.490350   1.094734  -0.717418  -5.179935     ...      0.621622
8842     -4.104215   0.553934  -1.498468  -4.594952     ...      0.573898
149874   -0.248337   1.091157  -0.307137  -5.567947     ...      0.450381
189701   -3.767189   0.061466  -1.836200  -1.470645     ...      1.581480
6971      5.431271  -1.946734  -0.775680  -1.987773     ...     -1.052368
6820     -4.235253   1.703538  -1.305279  -6.716720     ...      0.645103
79525    -3.207076  -1.482583  -5.074871  -6.778331     ...     -0.527474
192687    0.757011  -0.444418   0.997921  -1.429490     ...      0.019649
...            ...        ...        ...        ...     ...           ...
10337    -0.099481   0.120222   1.442236  -0.254566     ...      0.081068
115396    0.806295  -0.183403   0.495209  -2.070952     ...     -0.153038
88347     0.652338   0.350483  -0.849458  -0.446474     ...      0.185438
243657   -1.215335  -0.238649  -1.071280   1.545176     ...     -0.136977
274660   -0.322067   0.377306   1.080894  -0.697659     ...      0.044410
214155    0.791186   0.090527   0.883564   1.245114     ...     -0.436503
```

```
240014  -1.111033   0.034118   0.374068   0.767558    ...      0.295911
59100   -0.533197   0.180583   0.317044   0.346131    ...     -0.018152
241384   0.199361  -0.380579   0.652795   0.014350    ...      0.254185
206068  -0.386766   1.145076   0.374682  -0.810281    ...      0.334980
50249    0.586538  -0.284289  -1.001115   0.735356    ...      0.022452
248184   0.627464   0.824239  -0.839613  -0.549966    ...      0.044309
233242  -0.879575   1.263380  -1.644238   0.595432    ...      0.215538
157333  -0.075321   0.365748   1.598138  -1.117891    ...      0.046862
202540  -0.735044   0.145132   1.337449  -0.241916    ...      0.209496
173688  -0.789038   0.203621   1.388542   0.021249    ...      0.193535
26359   -0.376067   0.747415  -0.687990  -1.022811    ...      0.486000
216148  -9.999009   2.103347  -0.403108   0.065124    ...      2.210891
94662   -0.681225   0.847104  -0.529536  -0.363689    ...      0.205810
172899  -1.087857   0.172542  -0.095392   0.865382    ...      0.127523
21933   -0.066809  -0.018813  -2.750991   0.884114    ...     -0.184506
112627   0.144480   0.407072  -1.171796  -0.455056    ...     -0.093288
216440   0.135889  -0.220195   0.394268   0.189238    ...      0.264280
141899  -0.351366   0.285652   1.027651  -1.386219    ...      0.053922
212511  -0.112846   0.875467   0.120017  -0.356916    ...      0.437306
220937   1.325552  -0.091259  -0.926392  -0.625039    ...     -0.106016
4071    -0.008438   0.621245   0.804813  -0.638447    ...     -0.323484
213129  -0.410052  -0.105912  -0.551470   0.815114    ...     -0.192935
242789   0.324136  -0.220288   0.450874  -0.355476    ...      0.303838
124375   0.478319   0.020356  -0.817320  -0.478497    ...     -0.129579

              V22        V23       V24       V25       V26       V27       V28  \
14104   -0.889597  -0.074208  0.035446  0.550578 -0.027171 -0.024921  0.073605
43061   -0.832074  -0.186117  0.429781  0.697103  0.056031 -1.310888 -0.707403
74507    0.610479   0.789023 -0.564512  0.201196 -0.111225  1.144599  0.102280
276071   0.831939   0.142007  0.592615 -0.196143 -0.136676  0.020182 -0.015470
42473    0.620749  -0.094069  0.536719  0.398142  0.008277  2.053524  0.835749
184379  -0.531496  -0.328741  0.393100  0.568435  0.786605 -0.146102  0.076211
234632  -0.425938  -0.154440 -0.018820  0.632234  0.192922  0.468181  0.280486
143335  -0.451086   0.127214 -0.339450  0.394096  1.075295  1.649906 -0.394905
10690   -1.504119 -19.254328  0.544867 -4.781606 -0.007772  3.052358 -0.775036
12108    0.972755   1.241866 -1.051086  0.038009  0.672317  2.108471 -1.421243
30100   -0.890421  -0.325814  0.123040 -0.093014  0.232106 -0.310519 -0.745295
27749    0.090356  -0.341881 -0.215924  1.053032  0.271139  1.373300  0.691195
281674  -0.295135  -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309
76555    1.092437   0.320133 -0.434643 -0.380687  0.213630  0.423620 -0.105169
191359  -0.431790  -0.092088  0.145216  0.457788  0.167739  0.451243  0.268421
108258   0.189311  -0.005524 -0.814708  0.400924  0.286281  0.135215  0.257315
17407   -2.175815  -1.365104  0.174286  2.103868 -0.209944  1.278681  0.372393
152019   2.941475   0.916236 -0.255504 -0.183835 -0.584539 -0.315484 -0.097223
191267   1.018191   0.303550  0.833886 -1.222306  2.745261 -0.220402  0.168233
238222  -0.907720  -0.680108 -0.349170  0.056276 -1.149923 -1.809886  0.723051
6899    -0.435901  -0.384766 -0.286016  1.007934  0.413196  0.280284  0.303937
154587   8.361985   4.909111  0.098328 -1.508739  0.008711 -5.407824 -0.183811
```

```
208651  0.043807   0.102711 -0.601505  0.127371 -0.163009  0.853792  0.356503
8842   -0.080163   0.318408 -0.245862  0.338238  0.032271 -1.508458  0.608075
149874  0.521162   0.308325 -0.318012 -1.255362 -0.691963  0.264878 -0.130445
189701  0.261333   0.621415  0.994110 -0.687853 -0.337531 -1.612791  1.231425
6971    0.204817  -2.119007  0.170279 -0.393844  0.296367  1.985913 -0.900452
6820   -0.503529  -0.000523  0.071696  0.092007  0.308498  0.552591  0.298954
79525   0.220546  -1.371110 -0.504899  0.382307  0.395528  0.782036  0.628528
192687 -0.211678  -0.247452 -0.279472  0.239646 -0.508398 -0.015551  0.041881
...          ...        ...       ...       ...       ...       ...       ...
10337   0.747183  -0.200801 -0.346323 -0.333640 -0.245866 -0.062376 -0.084405
115396 -0.602683  -0.341616 -0.582677  0.767774 -0.308615 -0.036905  0.047589
88347   0.187364   0.181686  0.523048 -0.264937  0.232183 -0.160878 -0.039342
243657  0.142011   0.253495  0.033611 -0.278441 -0.180198  0.027348 -0.042797
274660  0.614156  -0.067753 -0.057593 -0.453118 -0.507858  0.297881  0.205687
214155 -0.713966   0.062884 -0.407804 -0.358156  0.110613  0.196411 -0.165517
240014  0.770382   0.127614  0.469469 -0.152423 -0.102730 -0.010070 -0.042385
59100   0.376450  -0.249901 -0.375701  0.773009  0.260953  0.056089  0.021437
241384  0.901867  -0.006258  0.011751  0.259469 -0.099717 -0.016691 -0.058387
206068  1.069969  -0.249956  0.742943 -0.249558 -0.102334  0.231554  0.172257
50249  -0.064824  -0.090029  0.697745  0.654846  0.046429 -0.049603  0.018543
248184  0.271363  -0.240952  0.589012  0.088004 -0.356156  0.465580  0.275718
233242  0.454978   0.180103  0.643387 -1.424133 -0.082591 -0.461710 -0.047992
157333  0.137862  -0.061153  0.559491 -0.174222 -1.040366 -0.054166  0.221529
202540  0.603047   0.066348 -0.040420 -0.278901  0.699988 -0.054265 -0.054437
173688  0.749558   0.033098 -0.799409 -0.073442 -0.179182  0.034539 -0.056260
26359  -0.006674   0.566694 -0.488192  0.590834 -0.344679 -0.140901 -0.301275
216148 -1.327692  -0.391332 -1.437327  0.649669 -0.169360  0.229878 -3.319224
94662   1.442508   2.693263 -0.321330  0.974952  0.078320  0.361779 -0.365369
172899 -0.005129   0.247272  0.183255 -0.538018 -0.467283 -0.017345 -0.028314
21933  -0.447778   0.052492 -0.056107 -0.022602 -0.480078  0.059544  0.101503
112627 -0.334119  -0.017765  0.429308 -0.160130  0.388421 -0.044990  0.016348
216440  0.886726  -0.070417 -0.459723  0.295745 -0.098154 -0.025310 -0.073800
141899  0.313926  -0.324390 -0.730735  0.342679 -0.424849  0.163478  0.089144
212511  1.008284  -0.065724  0.231887 -0.249200 -0.297614  0.077820  0.092511
220937 -0.393019  -0.325442 -0.384987  1.021907 -0.741675 -0.123620  0.059302
4071   -0.735670   0.111111  0.076782 -0.277719  0.043727  0.206915  0.073265
213129 -0.055736   0.158535  0.703399 -0.061192 -0.335567  0.013403 -0.035047
242789  0.625758  -0.222799 -0.567336  0.205051 -0.105940 -0.041244 -0.020069
124375 -0.436427  -0.045575  0.405231 -0.061127  0.236574  0.039432  0.078443

        Class  s_Amount
14104       1 -0.340755
43061       1 -0.216815
74507       1  0.168281
276071      1 -0.273468
42473       1 -0.338077
184379      1 -0.253277
234632      1 -0.346872
```

```
143335      1  0.657967
10690       1  4.519998
12108       1 -0.349231
30100       1 -0.110945
27749       1 -0.277186
281674      1 -0.183191
76555       1  0.260317
191359      1 -0.333759
108258      1 -0.350191
17407       1  0.046539
152019      1 -0.349231
191267      1 -0.324523
238222      1 -0.348831
6899        1 -0.349231
154587      1 -0.353229
208651      1 -0.195505
8842        1 -0.353229
149874      1  2.048541
189701      1  0.019992
6971        1  6.882027
6820        1 -0.349231
79525       1 -0.349231
192687      1  0.750922
...       ...       ...
10337       0  0.042581
115396      0 -0.323684
88347       0 -0.012393
243657      0 -0.269270
274660      0 -0.215695
214155      0 -0.317646
240014      0 -0.193346
59100       0 -0.347152
241384      0 -0.349231
206068      0 -0.339996
50249       0 -0.195385
248184      0 -0.349231
233242      0 -0.293258
157333      0 -0.259274
202540      0 -0.113344
173688      0 -0.313289
26359       0  1.169242
216148      0  0.165322
94662       0  0.153168
172899      0  0.146531
21933       0 -0.077361
112627      0 -0.342555
216440      0 -0.342475
141899      0 -0.349231
```

```
212511      0 -0.021228
220937      0 -0.261873
4071        0 -0.345313
213129      0 -0.053373
242789      0  0.293101
124375      0 -0.345313

[846 rows x 30 columns]
```

In [20]:
```
from collections import Counter #is a container that keeps track of how many times equi
Counter(under_sample_df['Class'])
```

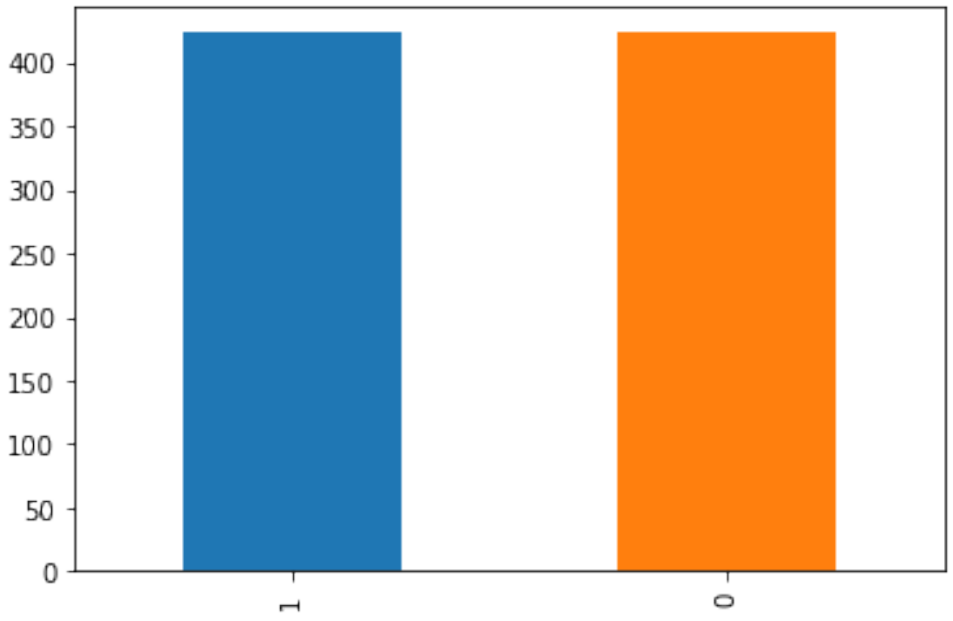
Out[20]:
```
Counter({0: 423, 1: 423})
```

In [21]:
```
under_sample_class_counts = pd.value_counts(under_sample_df['Class'])
under_sample_class_counts
```

Out[21]:
```
1    423
0    423
Name: Class, dtype: int64
```

In [22]:
```
under_sample_class_counts.plot(kind='bar')
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28b2213c0b8>
```



In [23]:
```
#Split the data into x and y variables
x_train_under = under_sample_df.loc[:, under_sample_df.columns!='Class']
y_train_under = under_sample_df.loc[:, under_sample_df.columns=='Class']
```

0.5.2 Oversampling of the dataset

```
In [24]: #2. Find the indices of the majority class
         index_non_fraud = y_train[y_train['Class']==0].index  #LISTA CON LAS POSICIONES
         #.3 Find the indices of the minority class
         index_fraud = y_train[y_train['Class']==1].index
```

```
In [25]: random_indices = np.random.choice(index_fraud, number_non_fraud,replace='False')
         len(random_indices)
```

```
Out[25]: 241662
```

```
In [26]: over_sample_indices = np.concatenate([index_non_fraud,random_indices])
         len(over_sample_indices)/2
```

```
Out[26]: 241662.0
```

```
In [27]: over_sample_df = data.iloc[over_sample_indices] #Purely integer-location based indexing
         over_sample_df
```

Out[27]:

	V1	V2	V3	V4	V5	V6	\
236135	-1.150681	-0.711104	1.396680	-3.141843	-0.798787	-0.062402	
168024	1.990980	-0.336590	-0.397329	0.391255	-0.430089	-0.131670	
234920	0.779102	-2.193339	-1.143066	2.151726	-0.658400	0.717706	
166991	1.796036	-0.355417	-0.812415	0.851164	-0.512741	-0.709853	
10637	1.076256	0.250711	0.502916	1.246392	0.136684	0.438216	
47380	-0.591400	0.745575	1.353398	-0.503739	0.237141	-0.688598	
146646	1.979947	-0.471303	-0.568579	0.334421	-0.375331	-0.018563	
145782	-1.169731	1.151179	0.640174	-0.714987	1.154392	1.145474	
280760	-0.262007	1.200465	-0.708319	-0.590111	0.591843	-0.552149	
269619	1.947831	-0.423775	-0.260640	0.265721	-0.858515	-0.614998	
45137	1.093537	-0.138962	1.376222	1.285632	-1.103832	-0.019828	
180111	-1.571283	-0.488801	-1.286119	-3.089391	0.248666	4.802101	
178217	-0.986292	0.801799	-0.147947	-0.711524	0.522296	-0.670951	
87797	-6.493504	-7.916078	1.766599	3.691802	8.601778	-5.285099	
160994	-1.210433	0.507319	1.328615	-3.404619	0.626261	-1.339666	
147644	2.009369	-0.274230	-1.307178	0.260487	0.033463	-0.508798	
217494	-0.287086	0.140884	1.901083	-0.896159	-1.005249	-0.923930	
103552	1.297847	0.538493	-0.619520	1.129266	0.303943	-0.752876	
230552	1.934579	-0.332531	-1.679585	0.207111	0.305872	-0.700871	
57969	1.315404	-0.033577	-1.089984	-0.557087	2.029931	3.268130	
3576	1.208863	0.273500	0.268125	0.648749	-0.218843	-0.610726	
269826	2.161113	-1.338818	-2.724866	-1.554723	-0.138538	-1.277205	
247394	-1.387686	0.107351	0.961606	-2.186933	-0.239950	-0.846542	
164795	2.084324	0.635847	-3.333443	0.505365	1.524477	-1.025286	
218399	0.194340	-1.077580	-1.130093	0.457347	-4.943108	2.438074	
21895	-1.142732	0.309447	1.348255	-1.231647	-0.073327	0.077899	
191676	1.948452	-0.118836	-2.215130	0.084623	1.459153	0.882699	
221931	2.230507	-0.656880	-2.404137	-1.192882	0.339214	-0.856712	

```
170971  -0.524847   1.001473   0.762588  -1.000115   0.429521 -0.280322
76912    1.359606  -0.453523  -0.184207  -0.721877  -0.454230 -0.204344
...           ...        ...        ...        ...        ...       ...
45732   -3.843009   3.375110  -5.492893   6.136378   2.797195 -2.646162
96341    1.227614  -0.668974  -0.271785  -0.589440  -0.604795 -0.350285
8972    -4.064005   3.100935  -1.188498   3.264633  -1.903562  0.320351
18472   -1.060676   2.608579  -2.971679   4.360089   3.738853 -2.728395
50537   -0.234922   0.355413   1.972183  -1.255593  -0.681387 -0.665732
151730  -1.952933   3.541385  -1.310561   5.955664  -1.003993  0.983049
79536   -0.264869   3.386140  -3.454997   4.367629   3.336060 -2.053918
15506  -21.885434  12.930505 -24.098872   6.203314 -16.466099 -4.459842
52521    1.001992   0.047938  -0.349002   1.493958   0.186939  0.190966
254395   0.202402   1.176270   0.346379   2.882138   1.407133 -0.504355
10801  -14.474437   6.503185 -17.712632  11.270352  -4.150142 -3.372098
68522    0.206075   1.387360  -1.045287   4.228686  -1.647549 -0.180897
53591   -1.309441   1.786495  -1.371070   1.214335  -0.336642 -1.390120
83053    0.326007   1.286638  -2.007181   2.419675  -1.532902 -1.432803
150660  -6.185857   7.102985 -13.030455   8.010823  -7.885237 -3.974550
52466   -1.476893   2.122314  -1.229470   1.201849  -0.343264 -1.317704
189878  -5.313774   2.664274  -4.250707   0.394707  -0.391383  0.683526
280149  -0.676143   1.126366  -2.213700   0.468308  -1.120541 -0.003346
203700   1.204934   3.238070  -6.010324   5.720847   1.548400 -2.321064
149145  -2.405580   3.738235  -2.317843   1.367442   0.394001  1.919938
248296  -0.613696   3.698772  -5.534941   5.620486   1.649263 -2.335145
167305  -6.677212   5.529299  -7.193275   6.081321  -1.636071  0.500610
18773    0.269614   3.549755  -5.810353   5.809370   1.538808 -2.269219
151008 -26.457745  16.497472 -30.177317   8.904157 -17.892600 -1.227904
208651   0.630579   1.183631  -5.066283   2.179903  -0.703376 -0.103614
195383   0.469750  -1.237555  -1.767341   4.833490  -0.268715 -0.512760
42741   -9.001351   6.613284 -12.423635   7.519929 -10.266255 -2.113208
42856  -11.682215   6.332882 -13.297109   7.690772 -10.889891 -2.792360
15539  -22.561699  13.208904 -24.643819   6.232532 -16.905611 -4.497439
147548  -3.859881   2.632881  -5.264265   3.446113  -0.675231 -1.904959

              V7         V8         V9        V10     ...          V21  \
236135  -0.477159   0.382365  -1.605210   1.126415     ...    -0.208415
168024  -0.581222  -0.027239   1.293546  -0.205487     ...     0.196103
234920   0.195957  -0.087316  -0.298813   0.806997     ...     0.458295
166991  -0.346712  -0.146682   0.938698  -0.511262     ...     0.259626
10637   -0.216999   0.155620   1.038624  -0.225015     ...     0.034257
47380    0.899851  -0.070925  -0.345476  -0.720276     ...     0.105988
146646  -0.589542  -0.011938   1.128638  -0.022798     ...     0.191475
145782   0.657856   0.612370  -0.108493  -0.814628     ...    -0.211771
280760   0.501543   0.494022  -0.449943  -0.894394     ...    -0.233417
269619  -0.662907   0.135912   1.326475  -0.004576     ...    -0.081138
45137   -0.714264   0.271273   0.746154  -0.008520     ...    -0.006514
180111   1.630077   0.993683  -1.499086  -0.924875     ...     0.289333
178217   0.518839   0.534855  -0.083745  -1.753175     ...    -0.087880
```

87797	-5.093447	0.681439	-0.327785	0.510737	...	0.784551
160994	1.524181	-0.744937	1.421291	-1.291796	...	-0.121756
147644	-0.044238	-0.009319	0.561100	0.264770	...	-0.215613
217494	0.319410	-0.204963	-1.330326	0.036423	...	-0.429643
103552	0.178193	-0.106125	0.201291	-0.520825	...	-0.124272
230552	0.383177	-0.317493	0.420803	0.028082	...	-0.163650
57969	-0.534994	0.808250	-0.052724	0.001149	...	-0.313307
3576	-0.012536	-0.045442	0.082681	-0.317258	...	-0.262837
269826	0.208608	-0.655256	-2.156480	1.668752	...	0.190213
247394	0.064976	0.428749	-1.297326	-0.451766	...	0.258910
164795	0.704292	-0.292717	-0.219665	-0.886690	...	-0.036584
218399	2.698650	-2.407959	-0.899390	0.295789	...	1.333232
21895	-0.350968	0.733975	-0.203022	-0.911345	...	0.276671
191676	0.314409	0.141746	0.147608	-0.024150	...	0.207921
221931	0.089435	-0.328408	-1.095334	1.109431	...	0.367909
170971	0.540743	-0.039551	1.038476	-0.694699	...	-0.358302
76912	-0.692101	0.030011	-0.608644	0.129876	...	0.029182
...	...	...	...	...	...	...
45732	-1.668931	-2.617552	-3.945843	-4.565252	...	-1.277812
96341	-0.486365	-0.010809	-0.794944	0.264545	...	-0.026055
8972	-0.954940	-3.277535	2.820829	1.015113	...	1.688665
18472	1.987616	-0.357345	-2.757535	-2.335933	...	-0.063168
50537	0.059110	-0.003153	1.122451	-1.481246	...	0.220670
151730	-4.587235	-4.892184	-2.516752	-2.850324	...	-1.998091
79536	0.256890	-2.957235	-2.855797	-2.808456	...	-1.394504
15506	-16.519836	14.535565	-3.897022	-8.650758	...	1.762232
52521	-0.001112	0.147140	0.580415	-0.792938	...	-0.334417
254395	1.438537	-0.395603	-1.555142	1.081514	...	0.242560
10801	-16.535807	-1.443947	-6.815273	-13.670545	...	-2.475962
68522	-2.943678	0.859156	-1.181743	-3.096504	...	0.469199
53591	-1.709109	0.667748	-1.699809	-3.843911	...	0.533521
83053	-2.459530	0.617738	-1.125861	-3.236784	...	0.556895
150660	-12.229608	4.971232	-4.248307	-12.965481	...	2.502772
52466	-1.528142	-0.620953	-1.213040	-2.975267	...	1.186036
189878	-5.133671	-7.907790	0.215475	-2.297734	...	8.664662
280149	-2.234739	1.210158	-0.652250	-3.463891	...	0.751826
203700	-0.781880	0.076619	-2.976249	-4.070257	...	0.098341
149145	-3.106942	-10.764403	3.353525	0.369936	...	10.005998
248296	-0.907188	0.706362	-3.747646	-4.230984	...	0.319261
167305	-4.640770	-4.339840	-0.950036	0.566680	...	5.563301
18773	-0.824203	0.351070	-3.759059	-4.592390	...	0.371121
151008	-31.197329	-11.438920	-9.462573	-22.187089	...	-8.755698
208651	-3.490350	1.094734	-0.717418	-5.179935	...	0.621622
195383	1.140149	-0.341273	-1.046351	0.085662	...	0.303905
42741	-9.984287	5.541941	-7.383705	-13.215172	...	1.775891
42856	-12.561783	7.287122	-7.570322	-12.835738	...	2.133456
15539	-16.810184	14.955107	-3.871297	-8.581266	...	1.765987
147548	-3.291041	-0.985766	-1.168114	-3.936294	...	1.664119

	V22	V23	V24	V25	V26	V27	V28	\
236135	-0.260557	-0.190488	0.591364	0.605121	-0.151481	0.361944	0.204409	
168024	0.839187	0.132476	0.762191	-0.030690	-0.248518	0.043946	-0.027562	
234920	-0.150650	-0.322586	0.227426	-0.870469	2.174588	-0.286915	0.049879	
166991	0.818557	0.004714	-0.029247	-0.192282	0.451567	-0.003003	-0.001525	
10637	0.382551	-0.060216	-0.337349	0.459101	-0.278657	0.016302	0.000954	
47380	0.160795	-0.259642	-0.110894	0.189140	0.319165	0.014191	0.092575	
146646	0.730220	0.100961	0.559064	-0.146435	0.591321	-0.027679	-0.039307	
145782	-0.380064	-0.231333	-0.387873	0.493134	-0.285715	-0.107481	-0.178727	
280760	-0.707854	0.175564	0.592964	-0.464226	0.095431	0.093125	0.011781	
269619	-0.215384	0.374069	-0.030208	-0.469471	-0.943435	0.044191	-0.039942	
45137	0.108226	0.017576	0.510228	0.328372	-0.435981	0.063366	0.028396	
180111	0.723290	-0.030687	0.771221	1.248623	0.098669	-0.033941	-0.177712	
178217	-0.467980	-0.166735	-0.560867	0.043783	-0.077247	-0.038145	0.040060	
87797	-0.085752	1.304263	-0.233903	0.963243	0.056030	-0.200075	0.250360	
160994	0.169537	-0.498999	0.020634	0.979029	-0.364106	-0.464313	-0.353622	
147644	-0.622877	0.324361	0.732148	-0.269722	0.225091	-0.094513	-0.068181	
217494	-0.738250	0.353038	0.923480	-0.974944	0.211384	0.142109	0.187939	
103552	-0.355117	-0.205121	-0.604849	0.753081	-0.270523	0.021067	0.039546	
230552	-0.560924	0.167063	0.509787	-0.123406	0.411452	-0.111754	-0.047341	
57969	-1.068934	0.103797	0.992126	0.369398	0.110374	-0.028302	0.011836	
3576	-0.745933	0.161312	0.024238	0.130420	0.125096	-0.009321	0.027005	
269826	0.787864	-0.240028	0.673879	0.632607	0.263798	-0.095336	-0.059414	
247394	0.388824	-0.254344	1.094421	0.862756	-0.159912	0.100090	-0.011285	
164795	0.076858	-0.095794	0.123166	0.408984	0.689850	-0.082362	-0.027284	
218399	-0.694768	0.959127	-0.072023	-0.512577	-0.165196	0.801488	0.138242	
21895	0.408629	-0.281664	-0.776058	-0.049952	1.009962	-0.118834	-0.036906	
191676	0.923350	0.128520	-0.533593	0.086251	0.809102	-0.046861	-0.093577	
221931	0.915875	-0.109627	0.322520	0.461992	0.054246	-0.097840	-0.086563	
170971	-0.723072	-0.044553	0.443002	-0.155367	-0.312056	0.083928	-0.048617	
76912	-0.027380	-0.091052	-0.864327	0.379023	-0.201584	0.037102	0.036976	
...	...	...	...	...	...	...	...	
45732	0.719652	0.451125	-0.258094	0.656129	0.556676	0.739383	-0.203050	
96341	-0.295255	-0.180459	-0.436539	0.494649	-0.283738	-0.001128	0.035075	
8972	-0.078845	0.193731	0.479496	-0.506603	-0.409863	-3.036271	-0.630605	
18472	-0.207385	-0.183261	-0.103679	0.896178	0.407387	-0.130918	0.192177	
50537	0.912107	-0.286338	0.451208	0.188315	-0.531846	0.123185	0.039581	
151730	1.133706	-0.041461	-0.215379	-0.865599	0.212545	0.532897	0.357892	
79536	-0.166029	-1.452081	-0.251815	1.243461	0.452787	0.132218	0.424599	
15506	-1.579055	-0.951043	0.134565	1.507110	-0.222671	1.527655	0.453699	
52521	-1.014315	-0.128427	-0.946242	0.456090	-0.453206	0.046627	0.064698	
254395	0.841230	-0.370157	-0.026012	0.491954	0.234576	-0.279788	-0.331933	
10801	0.342391	-3.564508	-0.818140	0.153408	0.755079	2.706566	-0.992916	
68522	0.344930	-0.203799	0.376640	0.715485	0.226003	0.628545	0.319918	
53591	-0.022180	-0.299556	-0.226416	0.364360	-0.475102	0.571426	0.293426	
83053	0.169776	-0.174357	0.308061	0.710996	-0.231030	0.580495	0.300984	
150660	0.481691	0.480958	0.360319	-0.293354	-0.199193	-0.203917	0.398927	

```
52466  -0.040215 -0.238930  0.110144  0.045418 -0.569232  0.481019 -0.047555
189878 -2.716383  0.483559  0.079235  0.311065  0.555544  0.176740  0.362907
280149  0.834108  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361
203700 -0.845866 -0.031228  0.421146  0.388361  0.056035  0.491828  0.340847
149145 -2.454964  1.684957  0.118263 -1.531380 -0.695308 -0.152502 -0.138866
248296 -0.471379 -0.075890 -0.667909 -0.642848  0.070600  0.488410  0.292345
167305 -1.608272  0.965322  0.163718  0.047531  0.466165  0.278547  1.471988
18773  -0.322290 -0.549856 -0.520629  1.378210  0.564714  0.553255  0.402400
151008  3.460893  0.896538  0.254836 -0.738097 -0.966564 -7.263482 -1.324884
208651  0.043807  0.102711 -0.601505  0.127371 -0.163009  0.853792  0.356503
195383 -0.647075 -0.373014  0.260801 -0.496566 -0.245973 -0.117858  0.144774
42741  -1.224758  0.082594  0.452089  0.463827 -0.296928  0.526506 -0.450890
42856  -1.271509 -0.035304  0.615054  0.349024 -0.428923 -0.694935 -0.818970
15539  -1.635517 -0.998317  0.138972  1.559350 -0.222125  1.504425  0.445920
147548  0.785075  0.068412  0.778961 -0.863166 -0.006810 -1.065734  1.773326

        Class  s_Amount
236135      0 -0.117342
168024      0 -0.313289
234920      0  2.276312
166991      0  0.046539
10637       0 -0.273308
47380       0 -0.208139
146646      0 -0.221692
145782      0 -0.329041
280760      0 -0.317327
269619      0 -0.349231
45137       0 -0.313289
180111      0  1.641816
178217      0 -0.190187
87797       0 -0.213296
160994      0 -0.312649
147644      0 -0.308091
217494      0 -0.133375
103552      0 -0.349231
230552      0  0.025949
57969       0 -0.345313
3576        0 -0.337757
269826      0  0.228412
247394      0 -0.153365
164795      0 -0.350191
218399      0  3.515279
21895       0 -0.241283
191676      0 -0.350191
221931      0 -0.213176
170971      0 -0.337837
76912       0 -0.214096
...       ...       ...
```

```
45732       1 -0.349231
96341       1  0.038623
8972        1  0.365067
18472       1 -0.350511
50537       1 -0.349231
151730      1 -0.277426
79536       1 -0.349231
15506       1  0.046539
52521       1  0.070528
254395      1 -0.322884
10801       1 -0.349231
68522       1 -0.350191
53591       1 -0.349231
83053       1 -0.328161
150660      1 -0.173715
52466       1 -0.349231
189878      1 -0.349231
280149      1 -0.041818
203700      1 -0.353229
149145      1 -0.325283
248296      1 -0.353229
167305      1  0.070128
18773       1 -0.350511
151008      1 -0.349231
208651      1 -0.195505
195383      1  2.538227
42741       1  1.482172
42856       1  0.338719
15539       1  0.046539
147548      1 -0.348512

[483324 rows x 30 columns]
```

```
In [28]: Counter(over_sample_df['Class'])
```

```
Out[28]: Counter({0: 241662, 1: 241662})
```

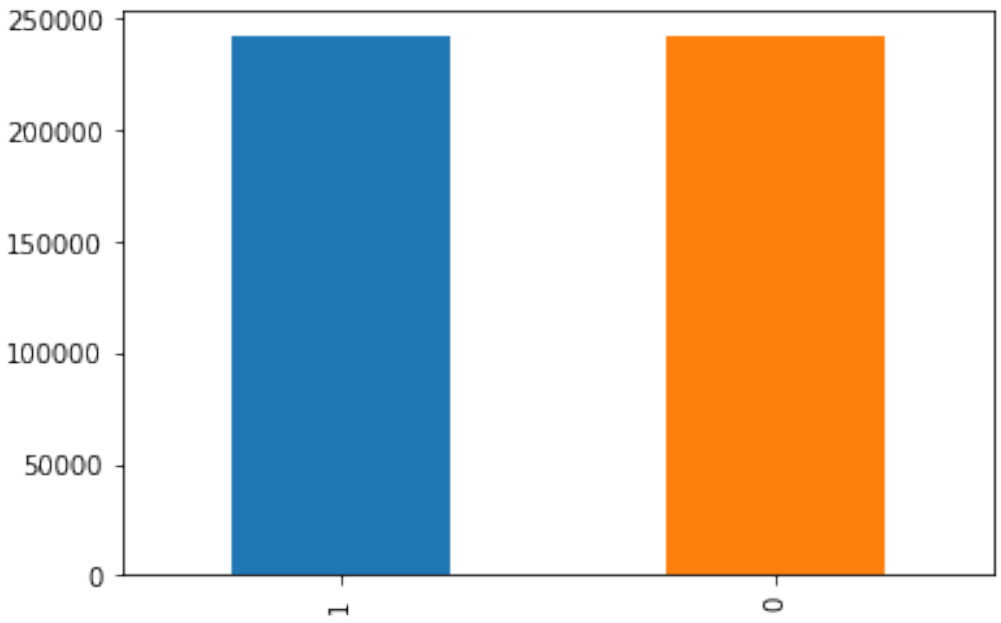
```
In [29]: #create a new dataframe containing only non-fraud data
         df_fraud = y_train[y_train['Class']==0]
         len(df_fraud)
```

```
Out[29]: 241662
```

```
In [30]: #Split the data into x and y variables
         x_train_over = over_sample_df.loc[:, over_sample_df.columns!='Class']
         y_train_over = over_sample_df.loc[:, over_sample_df.columns=='Class']
```

```
In [31]: over_sample_class_counts=pd.value_counts(over_sample_df['Class'])
         over_sample_class_counts
```

```
Out[31]: 1    241662
         0    241662
         Name: Class, dtype: int64

In [32]: over_sample_class_counts.plot(kind='bar')

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x28b223db898>
```



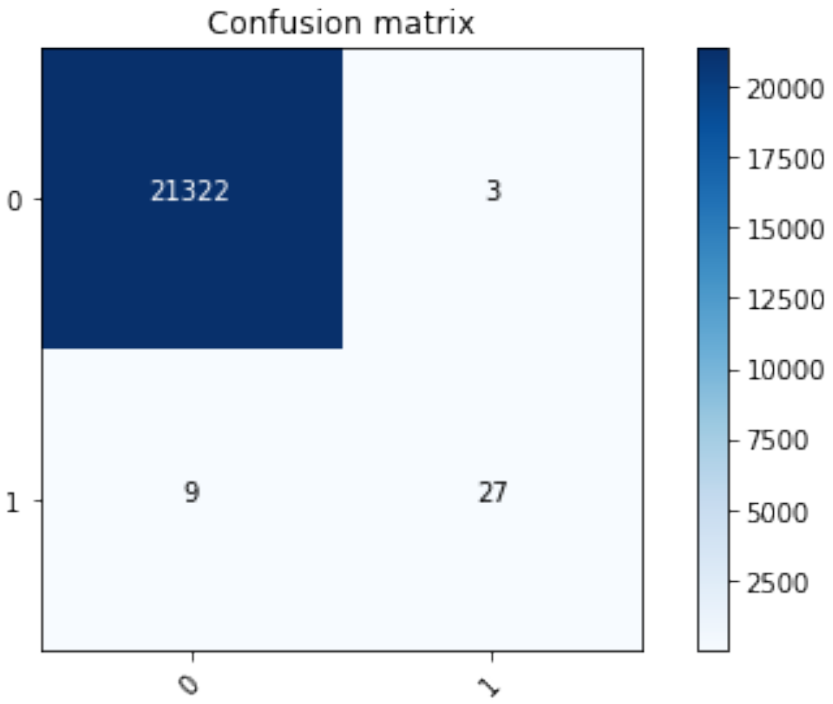
0.6 Model #1: Logistic Regression Classifier

Google's Scikit-learn implementation

i) Undersampling and oversampling study

```
In [33]: lr_normal = LogisticRegression()
         lr_normal.fit(x_train, y_train)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[33]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [34]: lr_normal.coef_ #Optimal Weights found by the model
```

```
Out[34]: array([[ 0.07778248, -0.0297105 ,  0.05436736,  0.61839087,  0.06443774,
                 -0.11237375, -0.14426912, -0.15483671, -0.39073887, -0.74840828,
                  0.01924601,  0.04590084, -0.27671627, -0.54861169, -0.07785509,
                 -0.19415309, -0.00638943,  0.00345472,  0.07789866, -0.43872281,
                  0.37567057,  0.59340809, -0.10843006,  0.00985883, -0.09867405,
                  0.15958119, -0.79219735, -0.30072612,  0.21568382]])

In [35]: lr_normal_predict = lr_normal.predict(x_val)
         print(lr_normal_predict[0:100]) #Sample of 100 predictions

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]


In [36]: #Performance measurement with different metrics (Original case)

         lr_normal_accuracy = accuracy_score(y_val, lr_normal_predict)
         lr_normal_recall = recall_score(y_val, lr_normal_predict)
         lr_normal_precision = precision_score(y_val, lr_normal_predict)
         lr_normal_f1score = f1_score(y_val, lr_normal_predict)
         lr_normal_roc_auc_score = roc_auc_score(y_val, lr_normal_predict)

         print("The Accuracy is", lr_normal_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_normal_precision)
         print("The Recall is",lr_normal_recall)
         print("The F1 Score is",lr_normal_f1score)
         print("ROCAUC:", lr_normal_roc_auc_score)

The Accuracy is 0.9994382285473526
The Precision is 0.9
The Recall is 0.75
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466


In [37]: cm = confusion_matrix(y_val, lr_normal_predict)
         plot_confusion_matrix(cm, class_names)
```




```
In [38]: lr_under = LogisticRegression() #Creates the model
         lr_under.fit(x_train_under, y_train_under) # Trains the model in the original training

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[38]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [39]: lr_under.coef_ #Optimal Weights found by the model

Out[39]: array([[ 2.08471145e-01,  2.98802182e-01,  1.60759060e-01,
                  7.19221101e-01,  2.76179569e-01, -5.67404410e-01,
                 -4.41912292e-01, -7.41836018e-01, -1.29839020e-01,
                 -4.21925393e-01,  4.66043685e-01, -6.63468154e-01,
                 -2.55773341e-01, -9.98394036e-01, -1.17361464e-01,
                 -4.75186817e-01, -3.12719918e-01, -1.29026595e-01,
                  2.58618355e-04, -1.15047877e-01,  4.58808551e-02,
                  4.76877724e-01,  3.02756971e-01, -2.93823515e-01,
                  2.48390212e-02, -1.70174417e-01,  5.14470821e-01,
                  1.47494366e-01,  1.11455436e+00]])
```

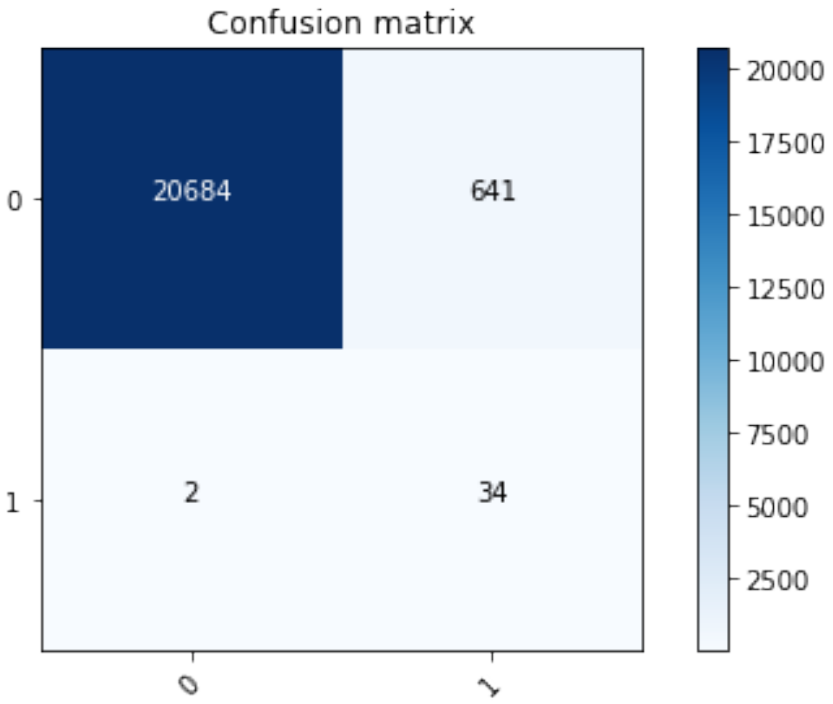
```
In [40]: lr_under_predict = lr_under.predict(x_val)
         print(lr_under_predict[0:100]) #Sample of 100 predictions
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
In [41]: #Performance measurement with different metrics (Undersampled case)

         lr_under_accuracy = accuracy_score(y_val, lr_under_predict)
         lr_under_recall = recall_score(y_val, lr_under_predict)
         lr_under_precision = precision_score(y_val, lr_under_predict)
         lr_under_f1score = f1_score(y_val, lr_under_predict)
         lr_under_roc_auc_score = roc_auc_score(y_val, lr_under_predict)

         print("The Accuracy is", lr_under_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_under_precision)
         print("The Recall is",lr_under_recall)
         print("The F1 Score is",lr_under_f1score)
         print("ROCAUC:", lr_under_roc_auc_score)
```

```
The Accuracy is 0.9698984129956463
The Precision is 0.05037037037037037
The Recall is 0.9444444444444444
The F1 Score is 0.09563994374120957
ROCAUC: 0.9571929138986583
```

```
In [42]: cm = confusion_matrix(y_val, lr_under_predict)
         plot_confusion_matrix(cm, class_names)
```




```
In [43]: lr_over = LogisticRegression()
         lr_over.fit(x_train_over, y_train_over)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[43]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [44]: lr_over.coef_ #Optimal Weights found by the model
```

```
Out[44]: array([[ 0.60826288,  0.56212867,  0.43442455,  0.75472986,  0.58481861,
                 -0.62682533, -0.59997958, -0.40038151, -0.37002965, -0.70800045,
                  0.55917028, -0.97956985, -0.32255691, -1.28882691, -0.06262105,
                 -0.68104086, -0.74871578, -0.2747559 ,  0.3065711 , -0.82840235,
                  0.07814055,  0.68519484,  0.40807121, -0.26935317,  0.12096079,
                 -0.17892668, -0.04520042,  0.85151523,  1.94139558]])
```

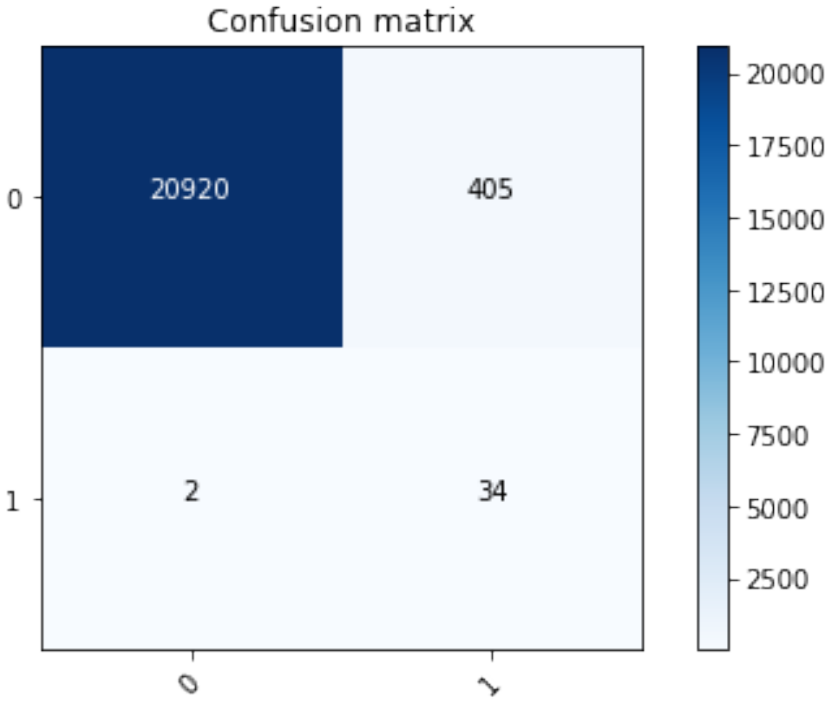
```
In [45]: lr_over_predict = lr_over.predict(x_val)
         print(lr_over_predict[0:100]) #Sample of 100 predictions
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

In [46]:
```
#Performance measurement with different metrics (Oversampled case)

lr_over_accuracy = accuracy_score(y_val, lr_over_predict)
lr_over_recall = recall_score(y_val, lr_over_predict)
lr_over_precision = precision_score(y_val, lr_over_predict)
lr_over_f1score = f1_score(y_val, lr_over_predict)
lr_over_roc_auc_score = roc_auc_score(y_val, lr_over_predict)

print("The Accuracy is", lr_over_accuracy) #acertar la prediccion en general
print("The Precision is", lr_over_precision)
print("The Recall is",lr_over_recall)
print("The F1 Score is",lr_over_f1score)
print("ROCAUC:", lr_over_roc_auc_score)
```

```
The Accuracy is 0.9809465848977108
The Precision is 0.0774487471526196
The Recall is 0.9444444444444444
The F1 Score is 0.14315789473684212
ROCAUC: 0.9627263253875212
```

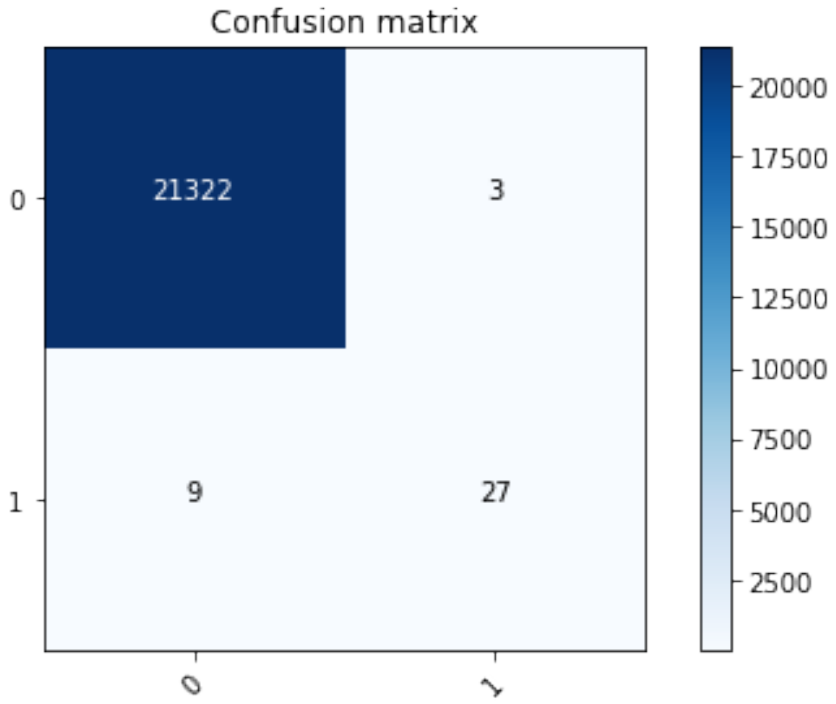
In [47]:
```
cm = confusion_matrix(y_val, lr_over_predict)
plot_confusion_matrix(cm, ['0', '1'] )
```



ii) Hyperparameter tuning Type of solver:

```
In [48]: tic = time.process_time()

         lr_normal = LogisticRegression(solver = 'newton-cg')
         lr_normal.fit(x_train, y_train)
         lr_normal_predict = lr_normal.predict(x_val)

         toc = time.process_time()

         time_newton_cg = toc - tic

         #Performance measurement with different metrics (newton-cg)

         lr_normal_accuracy = accuracy_score(lr_normal_predict, y_val)
         lr_normal_recall = recall_score(lr_normal_predict, y_val)
         lr_normal_precision = precision_score(lr_normal_predict, y_val)
         lr_normal_f1score = f1_score(lr_normal_predict,y_val)
         lr_normal_roc_auc_score = roc_auc_score(y_val, lr_normal_predict)

         print("The Accuracy is", lr_normal_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_normal_precision)
         print("The Recall is",lr_normal_recall)
         print("The F1 Score is",lr_normal_f1score)
         print("ROCAUC:", lr_normal_roc_auc_score)
         print("-------------")
         print("Computation time: ", time_newton_cg)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[48]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='newton-cg', tol=0.0001,
                   verbose=0, warm_start=False)

The Accuracy is 0.9994382285473526
The Precision is 0.75
The Recall is 0.9
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466
-------------
Computation time:  8.78125
```

In [49]:
```
cm = confusion_matrix(y_val, lr_normal_predict)

plot_confusion_matrix(cm, ['0', '1'] )
```



In [50]:
```
tic = time.process_time()

lr_normal_lfbgs = LogisticRegression(solver = 'lbfgs')
lr_normal_lfbgs.fit(x_train, y_train)
lr_normal_lfbgs_predict = lr_normal_lfbgs.predict(x_val)

toc = time.process_time()

time_lbfgs = toc - tic

#Performance measurement with different metrics (newton-cg)

lr_normal_lfbgs_accuracy = accuracy_score(lr_normal_lfbgs_predict, y_val)
lr_normal_lfbgs_recall = recall_score(lr_normal_lfbgs_predict, y_val)
lr_normal_lfbgs_precision = precision_score(lr_normal_lfbgs_predict, y_val)
lr_normal_lfbgs_f1score = f1_score(lr_normal_lfbgs_predict,y_val)
lr_normal_lfbgs_roc_auc_score = roc_auc_score(y_val, lr_normal_lfbgs_predict)

print("The Accuracy is", lr_normal_lfbgs_accuracy) #acertar la prediccion en general
print("The Precision is", lr_normal_lfbgs_precision)
```

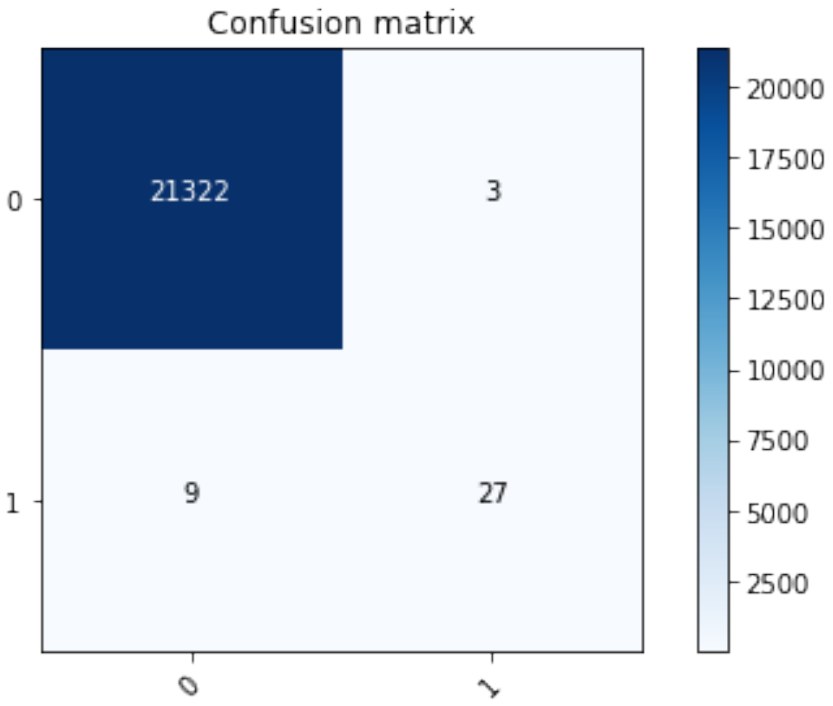
```
        print("The Recall is",lr_normal_lfbgs_recall)
        print("The F1 Score is",lr_normal_lfbgs_f1score)
        print("ROCAUC:", lr_normal_lfbgs_roc_auc_score)
        print("-------------")
        print("Computation time: ", time_lbfgs)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[50]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='lbfgs', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
The Accuracy is 0.9994382285473526
The Precision is 0.75
The Recall is 0.9
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466
-------------
Computation time:  6.71875
```

```
In [51]: cm = confusion_matrix(y_val, lr_normal_lfbgs_predict)

        plot_confusion_matrix(cm, ['0', '1'] )
```



```
In [52]: tic = time.process_time()

         lr_normal_sag = LogisticRegression(solver = 'sag')
         lr_normal_sag.fit(x_train, y_train)
         lr_normal_sag_predict = lr_normal_sag.predict(x_val)

         toc = time.process_time()

         time_sag = toc - tic

         #Performance measurement with different metrics (newton-cg)

         lr_normal_sag_accuracy = accuracy_score(y_val, lr_normal_sag_predict)
         lr_normal_sag_recall = recall_score(y_val, lr_normal_sag_predict)
         lr_normal_sag_precision = precision_score(y_val, lr_normal_sag_predict)
         lr_normal_sag_f1score = f1_score(y_val, lr_normal_sag_predict)
         lr_normal_sag_roc_auc_score = roc_auc_score(y_val, lr_normal_sag_predict)

         print("The Accuracy is", lr_normal_sag_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_normal_sag_precision)
         print("The Recall is", lr_normal_sag_recall)
         print("The F1 Score is",lr_normal_sag_f1score)
         print("ROCAUC:", lr_normal_sag_roc_auc_score)
         print("-------------")
         print("Computation time: ", time_sag)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning:
  "the coef_ did not converge", ConvergenceWarning)


Out[52]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='sag', tol=0.0001,
                    verbose=0, warm_start=False)

The Accuracy is 0.9994850428350732
The Precision is 0.9310344827586207
The Recall is 0.75
The F1 Score is 0.8307692307692308
ROCAUC: 0.8749531066822978
-------------
Computation time:  37.4375
```

```
In [53]: tic = time.process_time()

         lr_normal_saga = LogisticRegression(solver = 'saga')
         lr_normal_saga.fit(x_train, y_train)
         lr_normal_saga_predict = lr_normal_saga.predict(x_val)

         toc = time.process_time()

         time_saga = toc - tic

         #Performance measurement with different metrics (newton-cg)

         lr_normal_saga_accuracy = accuracy_score(y_val, lr_normal_saga_predict)
         lr_normal_saga_recall = recall_score(y_val, lr_normal_saga_predict)
         lr_normal_saga_precision = precision_score(y_val, lr_normal_saga_predict)
         lr_normal_saga_f1score = f1_score(y_val, lr_normal_saga_predict)
         lr_normal_saga_roc_auc_score = roc_auc_score(y_val, lr_normal_saga_predict)

         print("The Accuracy is", lr_normal_saga_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_normal_saga_precision)
         print("The Recall is",lr_normal_saga_recall)
         print("The F1 Score is",lr_normal_saga_f1score)
         print("ROCAUC:", lr_normal_saga_roc_auc_score)
         print("-------------")
         print("Computation time: ", time_saga)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning:
  "the coef_ did not converge", ConvergenceWarning)
```

```
Out[53]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='saga', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
The Accuracy is 0.9994850428350732
The Precision is 0.9310344827586207
The Recall is 0.75
The F1 Score is 0.8307692307692308
ROCAUC: 0.8749531066822978
-------------
Computation time:  38.796875
```

Regularization:

```
In [54]: lr_reg1 = LogisticRegression(penalty='l2', C=0.1)
         lr_reg1.fit(x_train, y_train)
```

```
        lr_reg1_predict = lr_reg1.predict(x_val)

        lr_reg1_accuracy = accuracy_score(y_val, lr_reg1_predict)
        lr_reg1_recall = recall_score(y_val, lr_reg1_predict)
        lr_reg1_precision = precision_score(y_val, lr_reg1_predict)
        lr_reg1_f1score = f1_score(y_val, lr_reg1_predict)
        lr_reg1_roc_auc_score = roc_auc_score(y_val, lr_reg1_predict)

        print("The Accuracy is", lr_reg1_accuracy) #acertar la prediccion en general
        print("The Precision is",lr_reg1_precision )
        print("The Recall is",lr_reg1_recall)
        print("The F1 Score is",lr_reg1_f1score)
        print("ROCAUC:", lr_reg1_roc_auc_score)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[54]: LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
The Accuracy is 0.9994850428350732
The Precision is 0.9310344827586207
The Recall is 0.75
The F1 Score is 0.8307692307692308
ROCAUC: 0.8749531066822978
```

```
In [55]: lr_reg1 = LogisticRegression(penalty='l2', C=0.01)
         lr_reg1.fit(x_train, y_train)
         lr_reg1_predict = lr_reg1.predict(x_val)

         lr_reg1_accuracy = accuracy_score(y_val, lr_reg1_predict)
         lr_reg1_recall = recall_score(y_val, lr_reg1_predict)
         lr_reg1_precision = precision_score(y_val, lr_reg1_predict)
         lr_reg1_f1score = f1_score(y_val, lr_reg1_predict)
         lr_reg1_roc_auc_score = roc_auc_score(y_val, lr_reg1_predict)

         print("The Accuracy is", lr_reg1_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_reg1_precision)
         print("The Recall is",lr_reg1_recall)
         print("The F1 Score is",lr_reg1_f1score)
         print("ROCAUC:", lr_reg1_roc_auc_score)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[55]: LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

The Accuracy is 0.9994382285473526
The Precision is 0.9
The Recall is 0.75
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466


In [56]: lr_reg1 = LogisticRegression(penalty='l2', C=10)
         lr_reg1.fit(x_train, y_train)
         lr_reg1_predict = lr_reg1.predict(x_val)

         lr_reg1_accuracy = accuracy_score(y_val, lr_reg1_predict)
         lr_reg1_recall = recall_score(y_val, lr_reg1_predict)
         lr_reg1_precision = precision_score(y_val, lr_reg1_predict)
         lr_reg1_f1score = f1_score(y_val, lr_reg1_predict)
         lr_reg1_roc_auc_score = roc_auc_score(y_val, lr_reg1_predict)

         print("The Accuracy is", lr_reg1_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_reg1_precision)
         print("The Recall is",lr_reg1_recall)
         print("The F1 Score is",lr_reg1_f1score)
         print("ROCAUC:", lr_reg1_roc_auc_score)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[56]: LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

The Accuracy is 0.9993914142596321
The Precision is 0.896551724137931
The Recall is 0.7222222222222222
The F1 Score is 0.7999999999999999
ROCAUC: 0.8610407711345578


In [57]: lr_reg1 = LogisticRegression(penalty='l2', C=3)
         lr_reg1.fit(x_train, y_train)
         lr_reg1_predict = lr_reg1.predict(x_val)

         lr_reg1_accuracy = accuracy_score(y_val, lr_reg1_predict)
```

```
        lr_reg1_recall = recall_score(y_val, lr_reg1_predict)
        lr_reg1_precision = precision_score(y_val, lr_reg1_predict)
        lr_reg1_f1score = f1_score(y_val, lr_reg1_predict)
        lr_reg1_roc_auc_score = roc_auc_score(y_val, lr_reg1_predict)

        print("The Accuracy is", lr_reg1_accuracy) #acertar la prediccion en general
        print("The Precision is", lr_reg1_precision)
        print("The Recall is",lr_reg1_recall)
        print("The F1 Score is",lr_reg1_f1score)
        print("ROCAUC:", lr_reg1_roc_auc_score)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[57]: LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
The Accuracy is 0.9994382285473526
The Precision is 0.9
The Recall is 0.75
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466
```

```
In [58]: lr_reg1 = LogisticRegression(penalty='l1', C=1)
         lr_reg1.fit(x_train, y_train)
         lr_reg1_predict = lr_reg1.predict(x_val)

         lr_reg1_accuracy = accuracy_score(y_val, lr_reg1_predict)
         lr_reg1_recall = recall_score(y_val, lr_reg1_predict)
         lr_reg1_precision = precision_score(y_val, lr_reg1_predict)
         lr_reg1_f1score = f1_score(y_val, lr_reg1_predict)
         lr_reg1_roc_auc_score = roc_auc_score(y_val, lr_reg1_predict)

         print("The Accuracy is", lr_reg1_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_reg1_precision)
         print("The Recall is",lr_reg1_recall)
         print("The F1 Score is",lr_reg1_f1score)
         print("ROCAUC:", lr_reg1_roc_auc_score)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[58]: LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
```

```
                   penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

The Accuracy is 0.9994382285473526
The Precision is 0.9
The Recall is 0.75
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466


In [59]: lr_reg1 = LogisticRegression(penalty='l1', C=0.1)
         lr_reg1.fit(x_train, y_train)
         lr_reg1_predict = lr_reg1.predict(x_val)

         lr_reg1_accuracy = accuracy_score(y_val, lr_reg1_predict)
         lr_reg1_recall = recall_score(y_val, lr_reg1_predict)
         lr_reg1_precision = precision_score(y_val, lr_reg1_predict)
         lr_reg1_f1score = f1_score(y_val, lr_reg1_predict)
         lr_reg1_roc_auc_score = roc_auc_score(y_val, lr_reg1_predict)

         print("The Accuracy is", lr_reg1_accuracy) #acertar la prediccion en general
         print("The Precision is", lr_reg1_precision)
         print("The Recall is",lr_reg1_recall)
         print("The F1 Score is",lr_reg1_f1score)
         print("ROCAUC:", lr_reg1_roc_auc_score)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[59]: LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

The Accuracy is 0.9994382285473526
The Precision is 0.9
The Recall is 0.75
The F1 Score is 0.8181818181818182
ROCAUC: 0.8749296600234466
```

iii) Tensorflow implementation

```
In [61]: !pip install tensorflow

Requirement already satisfied: tensorflow in c:\users\ruben\anaconda3\lib\site-packages
Requirement already satisfied: termcolor>=1.1.0 in c:\users\ruben\anaconda3\lib\site-packages (f
Requirement already satisfied: astor>=0.6.0 in c:\users\ruben\anaconda3\lib\site-packages (from
```

```
Requirement already satisfied: tensorboard<1.9.0,>=1.8.0 in c:\users\ruben\anaconda3\lib\site-pa
Requirement already satisfied: numpy>=1.13.3 in c:\users\ruben\anaconda3\lib\site-packages (from
Requirement already satisfied: wheel>=0.26 in c:\users\ruben\anaconda3\lib\site-packages (from t
Requirement already satisfied: gast>=0.2.0 in c:\users\ruben\anaconda3\lib\site-packages (from t
Requirement already satisfied: grpcio>=1.8.6 in c:\users\ruben\anaconda3\lib\site-packages (from
Requirement already satisfied: six>=1.10.0 in c:\users\ruben\anaconda3\lib\site-packages (from t
Requirement already satisfied: protobuf>=3.4.0 in c:\users\ruben\anaconda3\lib\site-packages (fr
Requirement already satisfied: absl-py>=0.1.6 in c:\users\ruben\anaconda3\lib\site-packages (fro
Requirement already satisfied: html5lib==0.9999999 in c:\users\ruben\anaconda3\lib\site-packages
Requirement already satisfied: bleach==1.5.0 in c:\users\ruben\anaconda3\lib\site-packages (from
Requirement already satisfied: werkzeug>=0.11.10 in c:\users\ruben\anaconda3\lib\site-packages (
Requirement already satisfied: markdown>=2.6.8 in c:\users\ruben\anaconda3\lib\site-packages (fr
Requirement already satisfied: setuptools in c:\users\ruben\anaconda3\lib\site-packages (from pr


You are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

```
In [62]: import tensorflow as tf
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the
  from ._conv import register_converters as _register_converters
```

```
In [63]: x_train.shape
```

```
Out[63]: (242085, 29)
```

```
In [64]: learning_rate = 0.01
         number_epochs = 300
         num_features = x_train.shape[1]

         tic_tf = time.process_time()

         tf.reset_default_graph()

         ##---------------------------##
         # Forward propagation
         ##---------------------------##

         X = tf.placeholder(tf.float32, [None, num_features], name="X")
         Y = tf.placeholder(tf.float32, [None, 1], name="Y")

         # Initialize our weigts & bias
         W = tf.get_variable("W", [num_features, 1], initializer = tf.zeros_initializer())
         b = tf.get_variable("b", [1], initializer = tf.zeros_initializer())

         Z = tf.add(tf.matmul(X, W), b)
```

```
prediction = tf.nn.sigmoid(Z)

# Calculate the cost
cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits = Z, labels = Y))

# Use Adam as optimization method
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()

cost_history = np.empty(shape=[1],dtype=float)


##---------------------------##
# Tensorflow Session
##---------------------------##

with tf.Session() as sess:

    sess.run(init)

    for epoch in range(number_epochs):
        _, c = sess.run([optimizer, cost], feed_dict={X: x_train, Y: y_train})
        #print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c))
        cost_history = np.append(cost_history, c)


    # Calculate the correct predictions
    correct_prediction = tf.to_float(tf.greater(prediction, 0.5))

    # Calculate accuracy on the test set
    accuracy = tf.reduce_mean(tf.to_float(tf.equal(Y, correct_prediction)))

    print ("Train Accuracy:", accuracy.eval({X: x_train, Y: y_train}))
    print ("Test Accuracy:", accuracy.eval({X: x_val, Y: y_val}))

    toc_tf = time.process_time()

    total_time_tf = toc_tf - tic_tf

    print("-------------")
    print("Computation time: ", total_time_tf)

    plt.figure(1, figsize=[8,5])
    plt.plot(cost_history[1:])
    plt.title("Learning curve of a Logistic classifier (Cost function vs. Number epochs
    plt.xlabel = "Number of Epochs"
    plt.ylabel = "Cost function"
```

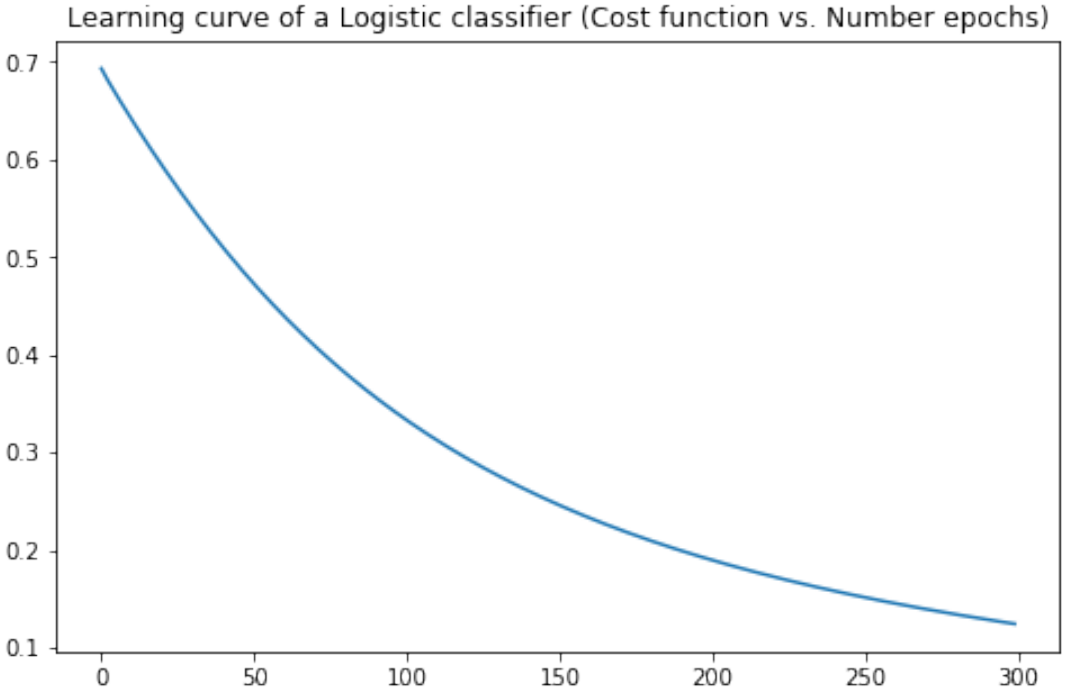
```
            plt.show

Train Accuracy: 0.99939275
Test Accuracy: 0.99953187
-------------
Computation time:  38.0625
```

Out[64]: <matplotlib.figure.Figure at 0x28b36de6f98>

Out[64]: [<matplotlib.lines.Line2D at 0x28b36e472e8>]

Out[64]: Text(0.5,1,'Learning curve of a Logistic classifier (Cost function vs. Number epochs)')

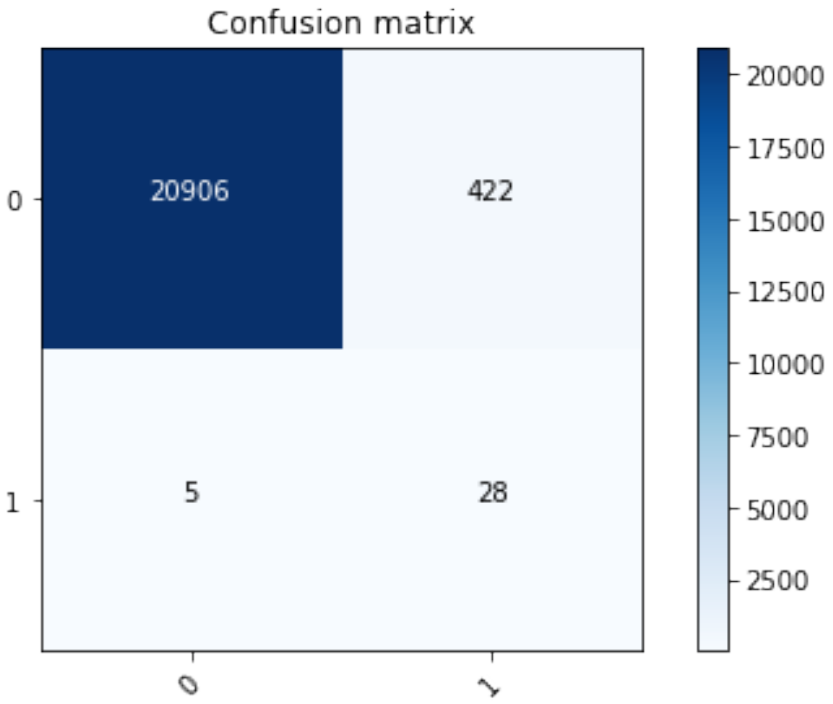
Out[64]: <function matplotlib.pyplot.show>



0.7 Apply model to Test Set: Oversampling

In [65]:
```
prediction_RL_test=lr_over.predict(x_test)
```

In [66]:
```
cm_test = confusion_matrix(y_test, prediction_RL_test)
plot_confusion_matrix(cm_test, class_names)
```




```
In [68]: #Performance measurement
         rl_model_accuracy = accuracy_score(y_test, prediction_RL_test)
         rl_model_recall = recall_score(y_test, prediction_RL_test)
         rl_model_precision = precision_score(y_test, prediction_RL_test)
         rl_model_f1score = f1_score(y_test, prediction_RL_test)
         rl_model_roc_auc_score = roc_auc_score(y_test, prediction_RL_test)

         print("The Accuracy is", rl_model_accuracy) #acertar la prediccion en general
         print("The Precision is", rl_model_precision)
         print("The Recall is",rl_model_recall)
         print("The F1 Score is",rl_model_f1score)
         print("ROCAUC:", rl_model_roc_auc_score)
```

```
The Accuracy is 0.9800102991432985
The Precision is 0.06222222222222222
The Recall is 0.8484848484848485
The F1 Score is 0.11594202898550725
ROCAUC: 0.9143493259678557
```

Anexo D

Código Support Vector Machines

En el siguiente anexo se detalla el código empleado para clasificar el conjunto de datos de estudio, mediante el algoritmo de *support vector machines*.

En el capítulo 5 se analizan los diferentes *outputs* que produce el siguiente código.

0.1 Import libraries

In [1]:
```
#Data processing
import numpy as np # biblioteca de funciones matemáticas de alto nivel para operar
                   #con esos vectores o matrices.
import pandas as pd # ofrece estructuras de datos y operaciones para manipular tablas nu
from sklearn.preprocessing import StandardScaler
from collections import Counter #is a container that keeps track of how many times equiv
import itertools

#General tools
import time
import sys

#Machine Learning Models
from sklearn import svm

#Metrics
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_
from sklearn.metrics import accuracy_score, recall_score, precision_score

#Plotting
import matplotlib #biblioteca para la generación de gráficos a partir de datos contenido
import matplotlib.pyplot as plt #plt.plot(x, y) implica coord y ord grafico
import seaborn as sns #Python visualization library based on matplotlib

%matplotlib inline
#guardar los graficos
```

In [2]:
```
from IPython.core.interactiveshell import InteractiveShell #que salgan todos los prints
InteractiveShell.ast_node_interactivity = "all"
```

0.2 Data loading

In [3]:
```
data = pd.read_csv("creditcard.csv") #subir los datos
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	\
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	

	V8	V9	...	V21	V22	V23	V24	\
0	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	
1	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	
2	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	
3	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	

```
4 -0.270533  0.817739  ...   -0.009431  0.798278 -0.137458  0.141267

        V25       V26       V27       V28  Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1  0.167170  0.125895 -0.008983  0.014724    2.69      0
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3  0.647376 -0.221929  0.062723  0.061458  123.50      0
4 -0.206010  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

0.3 Data visualization/analysis

```
In [4]: sc = StandardScaler() #Standardize features by removing the mean and scaling to unit var
        data['s_Amount']=sc.fit_transform(data['Amount'].values.reshape(-1,1)) #porque -1 1, lo

        data[['Amount', 's_Amount']].head()
```

```
Out[4]:    Amount  s_Amount
        0  149.62  0.244964
        1    2.69 -0.342475
        2  378.66  1.160686
        3  123.50  0.140534
        4   69.99 -0.073403
```

```
In [5]: data = data.drop(['Time', 'Amount'], axis=1) #borras las columnas
        data.columns
```

```
Out[5]: Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
               'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
               'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Class', 's_Amount'],
              dtype='object')
```

```
In [6]: class_names=['0','1'] # Binary label, Class = 1 (fraud) and Class = 0 (no fraud)

        def plot_confusion_matrix(cm, classes,
                                  title='Confusion matrix',
                                  cmap=plt.cm.Blues):

            plt.imshow(cm, interpolation='nearest', cmap=cmap)
            plt.title(title)
            plt.colorbar()
            tick_marks = np.arange(len(classes))
            plt.xticks(tick_marks, classes, rotation=45)
            plt.yticks(tick_marks, classes)

            fmt = 'd'
            thresh = cm.max() / 2.
            for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
```

0.4 Train/val/test set split

```
In [7]: #Split the data into x and y variables
        x = data.loc[:, data.columns != 'Class'] #pedir datos de la tabla, : implica todas las f
        y = data.loc[:, data.columns == 'Class']
        #separamos en inputs y outputs

In [8]: #Split the data into train and test data

        from sklearn.model_selection import train_test_split

        x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.15, random_state = 1

        x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.5, random_st

In [9]: print(len(x))
        print(len(x_train))
        print(len(x_test))
        print(len(x_val))
```

```
284807
242085
21361
21361
```

0.5 Model #2: Support Vector Machines

Google's Scikit-learn implementation

i) Hyperparameter study

- **Type of Kernel:**

```
In [10]: classifier = svm.SVC(kernel='linear') # We set a SVM classifier, the default SVM Classi
         classifierP = svm.SVC(kernel='poly')
         classifierR = svm.SVC(kernel='rbf')
         classifierS = svm.SVC(kernel='sigmoid')

In [11]: tic = time.process_time()
         classifier.fit(x_train[0:100000], y_train[0:100000])
```

```
        toc = time.process_time()

        time_linear = toc - tic
        print(time_linear)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[11]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
71.234375
```

```
In [12]: ticP = time.process_time()
         classifierP.fit(x_train[0:100000], y_train[0:100000])
         tocP = time.process_time()

         time_poly = tocP - ticP
         print(time_poly)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[12]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='poly',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
4.078125
```

```
In [13]: ticR = time.process_time()
         classifierR.fit(x_train[0:100000], y_train[0:100000])
         tocR = time.process_time()

         time_rbf = tocR - ticR
         print(time_rbf)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[13]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
29.71875
```

In [14]:
```
ticS = time.process_time()
classifierS.fit(x_train[0:100000], y_train[0:100000])
tocS = time.process_time()

time_sigmoid = tocS - ticS
print(time_sigmoid)
```

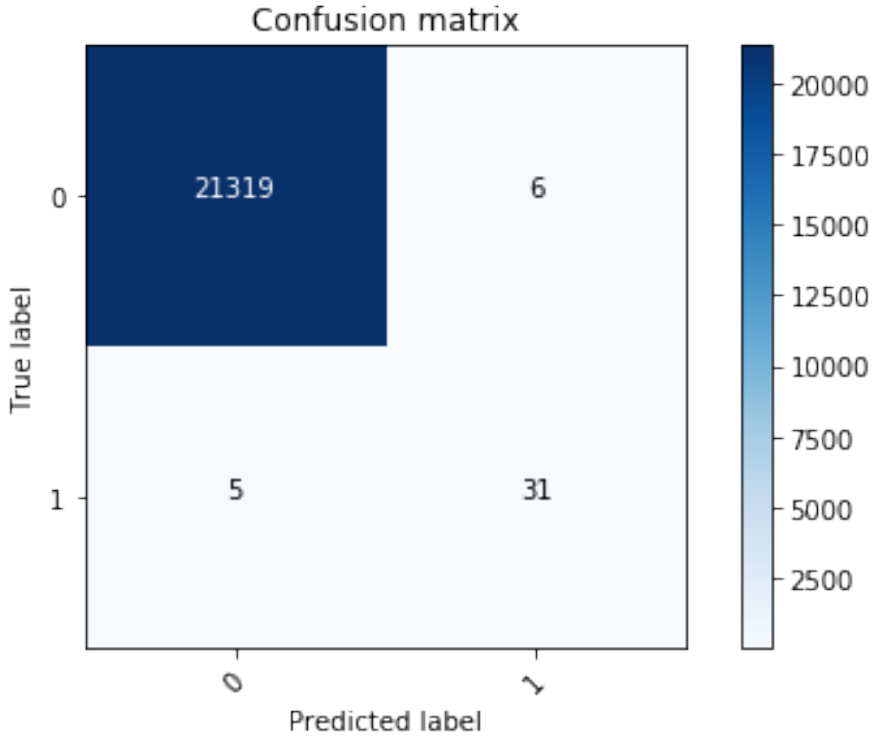
```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

Out[14]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='sigmoid',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

```
1.78125
```

In [15]:
```
prediction_SVM_linear = classifier.predict(x_val) #And finally, we predict our data tes
prediction_SVM_P = classifierP.predict(x_val)
prediction_SVM_R = classifierR.predict(x_val)
prediction_SVM_S = classifierS.predict(x_val)
```

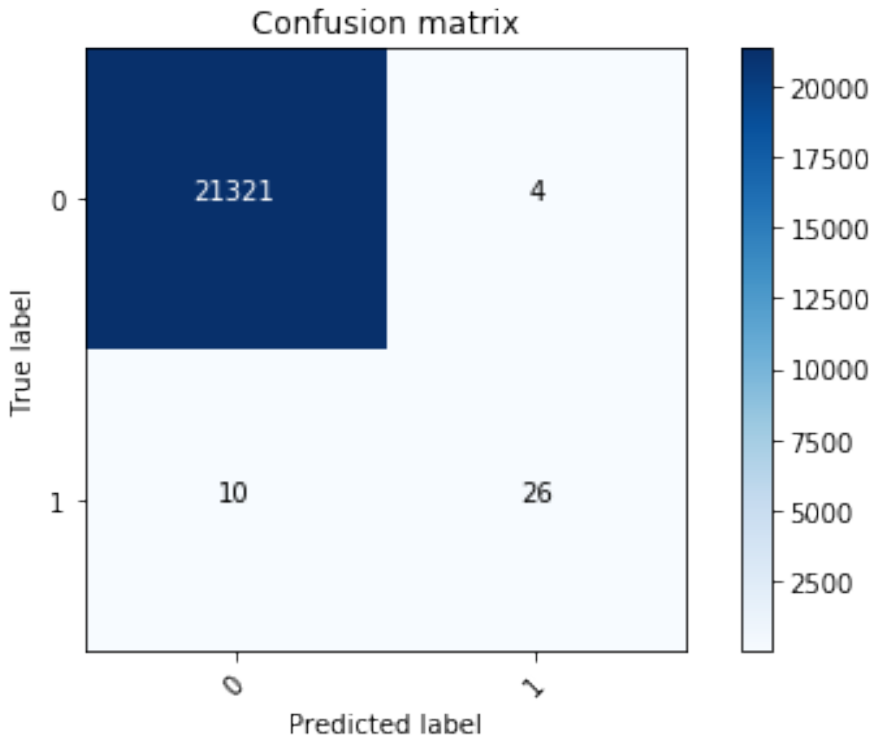
In [16]:
```
cm1 = confusion_matrix(y_val, prediction_SVM_linear)
plot_confusion_matrix(cm1, class_names)
```



In [17]:
```
#Performance measurement with different metrics (Linear kernel)

svm_linear_accuracy = accuracy_score(y_val, prediction_SVM_linear)
svm_linear_recall = recall_score(y_val, prediction_SVM_linear)
svm_linear_precision = precision_score(y_val, prediction_SVM_linear)
svm_linear_f1score = f1_score(y_val, prediction_SVM_linear)
svm_linear_roc_auc_score = roc_auc_score(y_val, prediction_SVM_linear)

print("The Accuracy is", svm_linear_accuracy) #acertar la prediccion en general
print("The Precision is", svm_linear_precision)
print("The Recall is",svm_linear_recall)
print("The F1 Score is",svm_linear_f1score)
print("ROCAUC:", svm_linear_roc_auc_score)
```

```
The Accuracy is 0.9994850428350732
The Precision is 0.8378378378378378
The Recall is 0.8611111111111112
The F1 Score is 0.8493150684931507
ROCAUC: 0.930414875602449
```

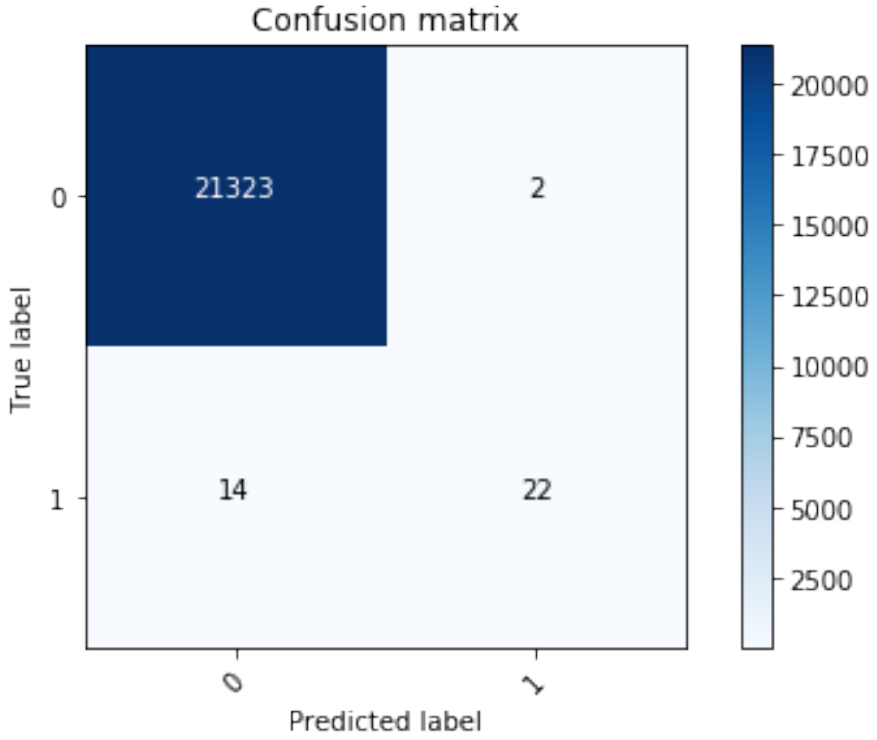
In [18]:
```
cm2 = confusion_matrix(y_val, prediction_SVM_P)
plot_confusion_matrix(cm2, class_names)
```



In [19]:
```
#Performance measurement with different metrics (Poly kernel)

svm_poly_accuracy = accuracy_score(y_val,prediction_SVM_P)
svm_poly_recall = recall_score(y_val, prediction_SVM_P)
svm_poly_precision = precision_score(y_val, prediction_SVM_P)
svm_poly_f1score = f1_score(y_val, prediction_SVM_P)
svm_poly_roc_auc_score = roc_auc_score(y_val, prediction_SVM_P)

print("The Accuracy is", svm_poly_accuracy) #acertar la prediccion en general
print("The Precision is", svm_poly_precision)
print("The Recall is",svm_poly_recall)
print("The F1 Score is",svm_poly_f1score)
print("ROCAUC:", svm_poly_roc_auc_score)
```

```
The Accuracy is 0.9993445999719114
The Precision is 0.8666666666666667
The Recall is 0.7222222222222222
The F1 Score is 0.7878787878787877
ROCAUC: 0.8610173244757068
```

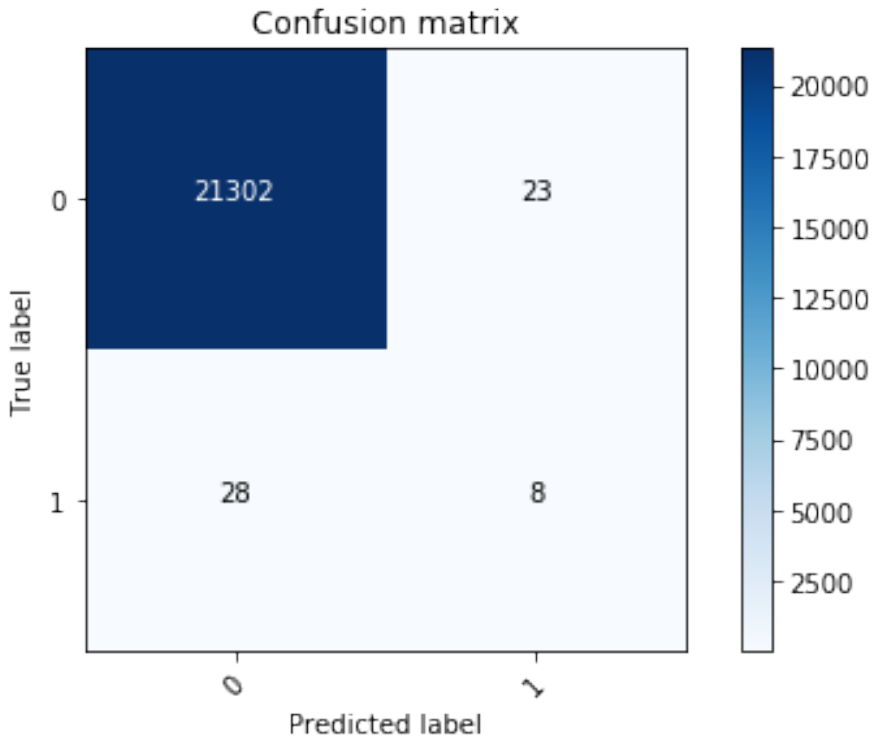
In [20]:
```
cm3 = confusion_matrix(y_val, prediction_SVM_R)
plot_confusion_matrix(cm3,class_names)
```



In [21]:
```
#Performance measurement with different metrics (Radial Basis Function kernel)

svm_rbf_accuracy = accuracy_score(y_val,prediction_SVM_R)
svm_rbf_recall = recall_score(y_val, prediction_SVM_R)
svm_rbf_precision = precision_score(y_val, prediction_SVM_R)
svm_rbf_f1score = f1_score(y_val, prediction_SVM_R)
svm_rbf_roc_auc_score = roc_auc_score(y_val, prediction_SVM_R)

print("The Accuracy is", svm_rbf_accuracy) #acertar la prediccion en general
print("The Precision is",svm_rbf_precision )
print("The Recall is",svm_rbf_recall)
print("The F1 Score is",svm_rbf_f1score)
print("ROCAUC:", svm_rbf_roc_auc_score)
```

```
The Accuracy is 0.9992509713964702
The Precision is 0.9166666666666666
The Recall is 0.6111111111111112
The F1 Score is 0.7333333333333334
ROCAUC: 0.8055086622378533
```

In [22]:
```
cm4 = confusion_matrix(y_val, prediction_SVM_S)
plot_confusion_matrix(cm4,class_names)
```



In [23]:
```
#Performance measurement with different metrics (Sigmoid kernel)

svm_sigmoid_accuracy = accuracy_score(y_val, prediction_SVM_S)
svm_sigmoid_recall = recall_score(y_val, prediction_SVM_S)
svm_sigmoid_precision = precision_score(y_val, prediction_SVM_S)
svm_sigmoid_f1score = f1_score(y_val, prediction_SVM_S)
svm_sigmoid_roc_auc_score = roc_auc_score(y_val, prediction_SVM_S)

print("The Accuracy is", svm_sigmoid_accuracy) #acertar la prediccion en general
print("The Precision is", svm_sigmoid_precision)
print("The Recall is",svm_sigmoid_recall)
print("The F1 Score is",svm_sigmoid_f1score)
print("ROCAUC:", svm_sigmoid_roc_auc_score)
```

```
The Accuracy is 0.9976124713262488
The Precision is 0.25806451612903225
The Recall is 0.2222222222222222
The F1 Score is 0.2388059701492537
ROCAUC: 0.6105718379575356
```

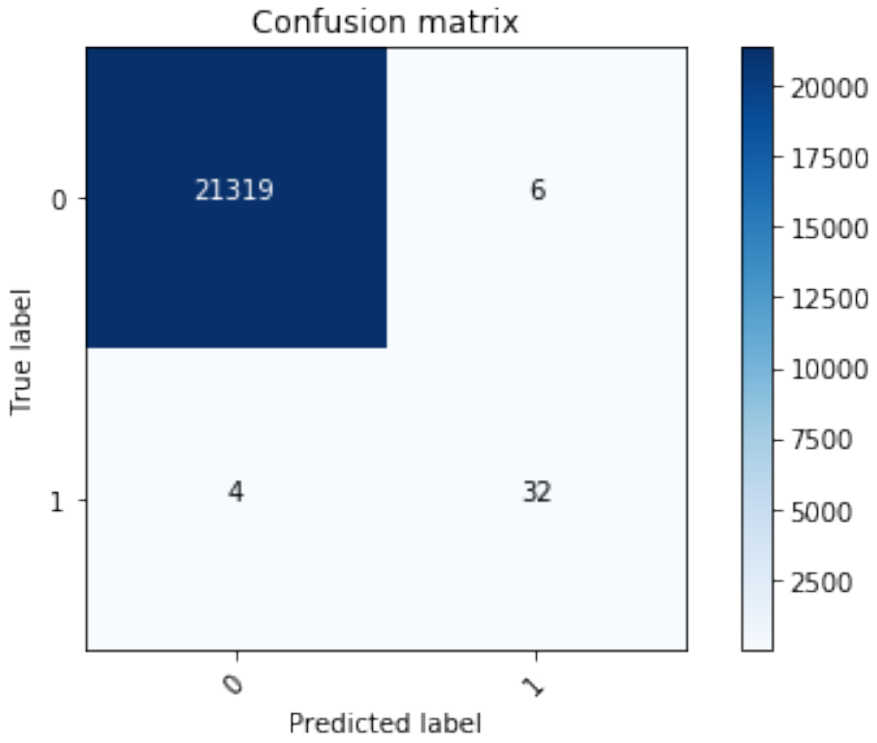
- **Degree of the polynomial SVM Kernel:**

In [24]:
```
classifierP1 = svm.SVC(kernel='poly', degree = 1)
classifierP1.fit(x_train[0:100000], y_train[0:100000])
prediction_SVM_P1 = classifierP1.predict(x_val)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

Out[24]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=1, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [25]:
```
cm5 = confusion_matrix(y_val, prediction_SVM_P1)
plot_confusion_matrix(cm5, class_names)
```



In [26]:
```
#Performance measurement with different metrics (Polynomial kernel degree 1)

svm_P1_accuracy = accuracy_score(y_val, prediction_SVM_P1)
svm_P1_recall = recall_score(y_val, prediction_SVM_P1)
svm_P1_precision = precision_score(y_val, prediction_SVM_P1)
svm_P1_f1score = f1_score(y_val, prediction_SVM_P1)
svm_P1_roc_auc_score = roc_auc_score(y_val, prediction_SVM_P1)
```

```
        print("The Accuracy is", svm_P1_accuracy) #acertar la prediccion en general
        print("The Precision is", svm_P1_precision)
        print("The Recall is",svm_P1_recall)
        print("The F1 Score is",svm_P1_f1score)
        print("ROCAUC:", svm_P1_roc_auc_score)
```

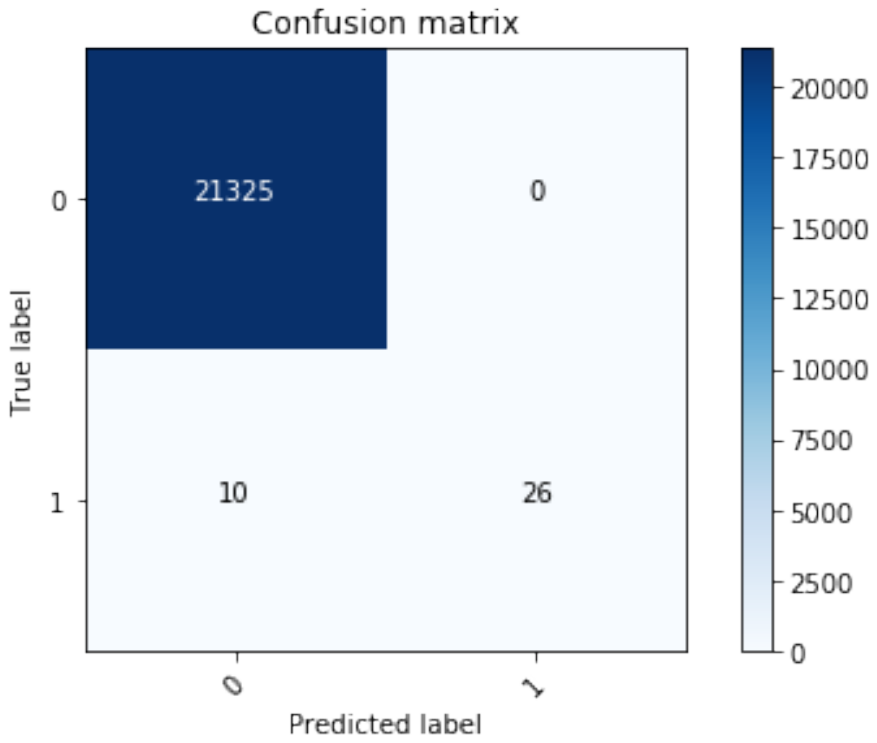
```
The Accuracy is 0.9995318571227939
The Precision is 0.8421052631578947
The Recall is 0.8888888888888888
The F1 Score is 0.8648648648648649
ROCAUC: 0.9443037644913378
```

```
In [27]: classifierP2 = svm.SVC(kernel='poly', degree = 2)
         classifierP2.fit(x_train[0:100000], y_train[0:100000])
         prediction_SVM_P2 = classifierP2.predict(x_val)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[27]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=2, gamma='auto', kernel='poly',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
In [28]: cm6 = confusion_matrix(y_val, prediction_SVM_P2)
         plot_confusion_matrix(cm6, class_names)
```



In [29]:
```
#Performance measurement with different metrics (Polynomial kernel degree 2)

svm_P2_accuracy = accuracy_score(y_val, prediction_SVM_P2)
svm_P2_recall = recall_score(y_val, prediction_SVM_P2)
svm_P2_precision = precision_score(y_val, prediction_SVM_P2)
svm_P2_f1score = f1_score(y_val, prediction_SVM_P2)
svm_P2_roc_auc_score = roc_auc_score(y_val, prediction_SVM_P2)

print("The Accuracy is", svm_P2_accuracy) #acertar la prediccion en general
print("The Precision is",svm_P2_precision )
print("The Recall is",svm_P2_recall)
print("The F1 Score is",svm_P2_f1score)
print("ROCAUC:", svm_P2_roc_auc_score)
```

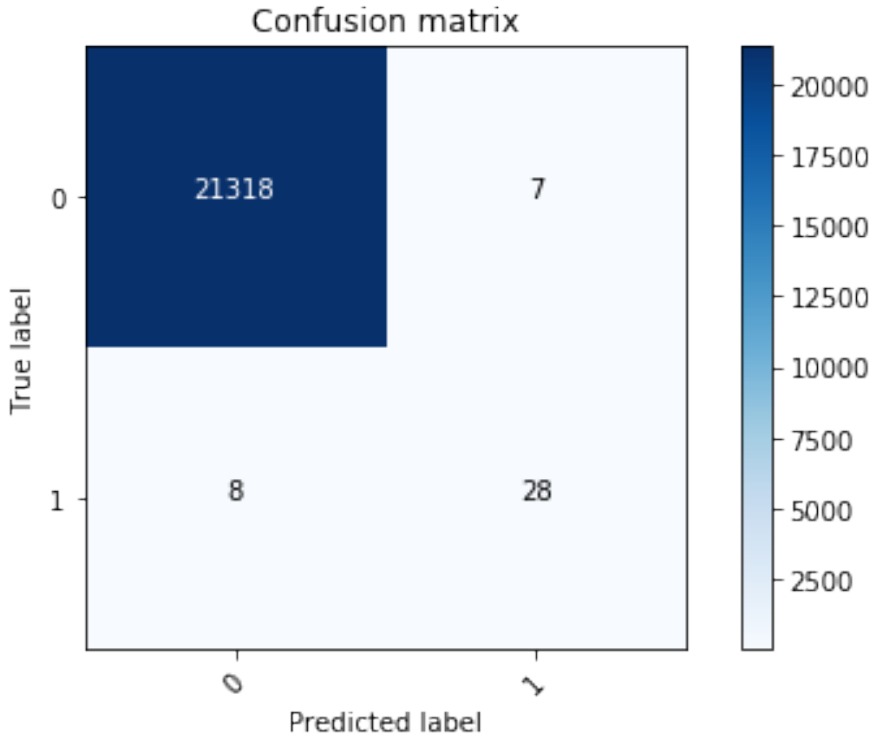
```
The Accuracy is 0.9995318571227939
The Precision is 1.0
The Recall is 0.7222222222222222
The F1 Score is 0.8387096774193548
ROCAUC: 0.8611111111111112
```

In [30]:
```
classifierP5 = svm.SVC(kernel='poly', degree = 5)
classifierP5.fit(x_train[0:100000], y_train[0:100000])
prediction_SVM_P5 = classifierP5.predict(x_val)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

Out[30]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=5, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [31]:
```
cm7 = confusion_matrix(y_val, prediction_SVM_P5)
plot_confusion_matrix(cm7, class_names)
```



In [32]:
```
#Performance measurement with different metrics (Polynomial kernel degree 5)

svm_P5_accuracy = accuracy_score(y_val, prediction_SVM_P5)
svm_P5_recall = recall_score(y_val, prediction_SVM_P5)
svm_P5_precision = precision_score(y_val, prediction_SVM_P5)
svm_P5_f1score = f1_score(y_val, prediction_SVM_P5)
svm_P5_roc_auc_score = roc_auc_score(y_val, prediction_SVM_P5)

print("The Accuracy is", svm_P5_accuracy) #acertar la prediccion en general
print("The Precision is", svm_P5_precision)
print("The Recall is",svm_P5_recall)
```

```
print("The F1 Score is",svm_P5_f1score)
print("ROCAUC:", svm_P5_roc_auc_score)
```

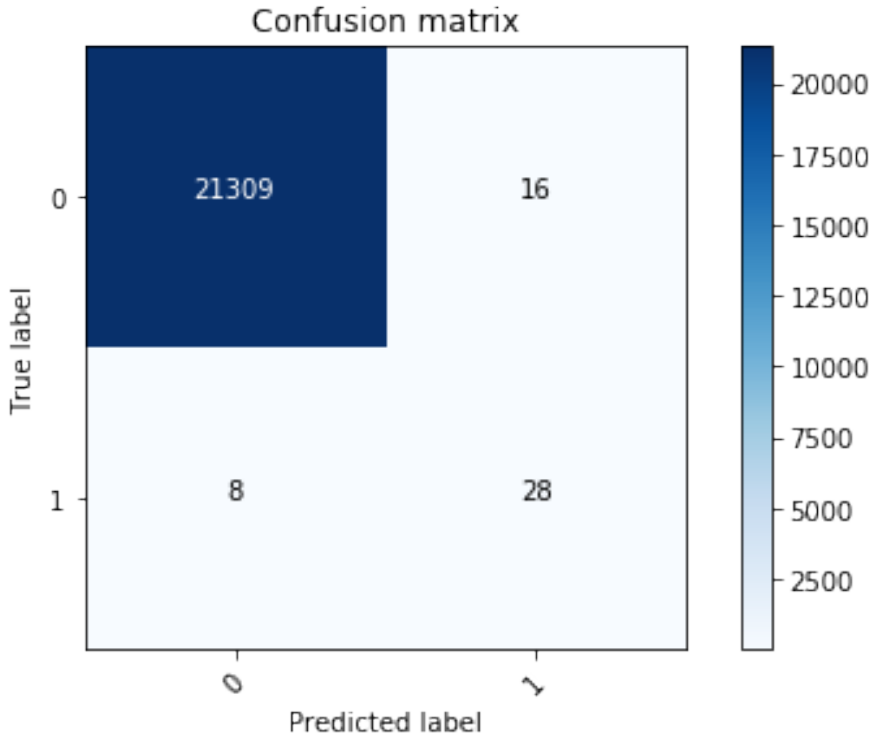
```
The Accuracy is 0.9992977856841908
The Precision is 0.8
The Recall is 0.7777777777777778
The F1 Score is 0.7887323943661971
ROCAUC: 0.888724762276931
```

```
In [33]: classifierP10 = svm.SVC(kernel='poly', degree = 10)
         classifierP10.fit(x_train[0:100000], y_train[0:100000])
         prediction_SVM_P10 = classifierP10.predict(x_val)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[33]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=10, gamma='auto', kernel='poly',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
In [34]: cm8 = confusion_matrix(y_val, prediction_SVM_P10)
         plot_confusion_matrix(cm8, class_names)
```



```
In [35]: #Performance measurement with different metrics (Polynomial kernel degree 10)

         svm_P10_accuracy = accuracy_score(y_val, prediction_SVM_P10)
         svm_P10_recall = recall_score(y_val, prediction_SVM_P10)
         svm_P10_precision = precision_score(y_val, prediction_SVM_P10)
         svm_P10_f1score = f1_score(y_val, prediction_SVM_P10)
         svm_P10_roc_auc_score = roc_auc_score(y_val, prediction_SVM_P10)

         print("The Accuracy is", svm_P10_accuracy) #acertar la prediccion en general
         print("The Precision is",svm_P10_precision )
         print("The Recall is",svm_P10_recall)
         print("The F1 Score is",svm_P10_f1score)
         print("ROCAUC:", svm_P10_roc_auc_score)
```

```
The Accuracy is 0.9988764570947053
The Precision is 0.6363636363636364
The Recall is 0.7777777777777778
The F1 Score is 0.7000000000000001
ROCAUC: 0.8885137423472711
```

- **Changing class weights:**

```
In [36]: classifier_w1 = svm.SVC(kernel='linear',class_weight={0:0, 1:1})
         classifier_w1.fit(x_train[0:10000], y_train[0:10000]) # Then we train our model, with o

         classifier_w2 = svm.SVC(kernel='linear',class_weight={0:1, 1:0})
         classifier_w2.fit(x_train[0:10000], y_train[0:10000]) # Then we train our model, with o
```

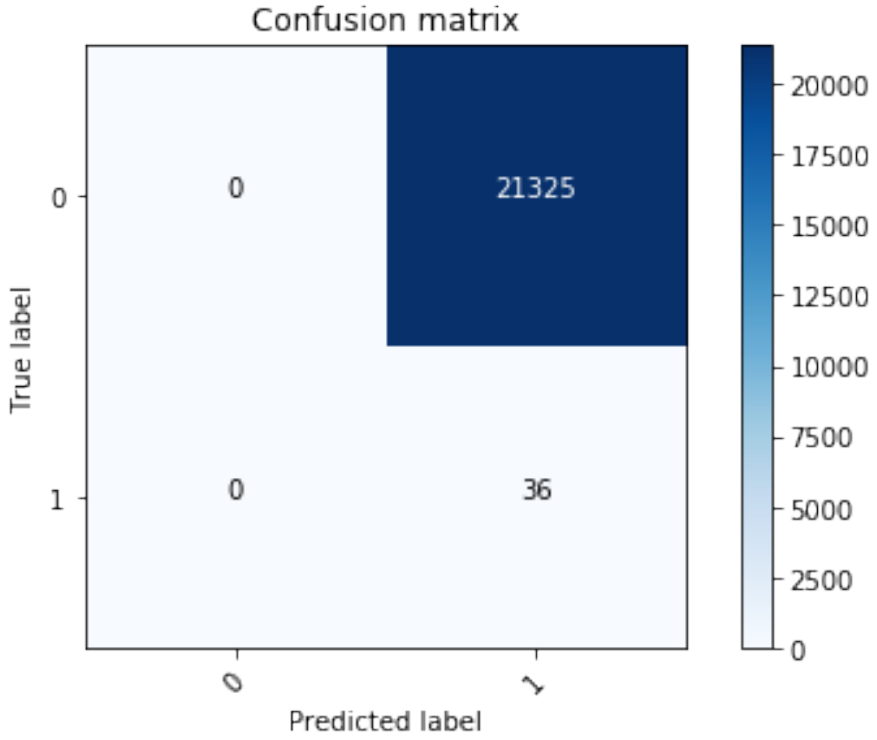
```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[36]: SVC(C=1.0, cache_size=200, class_weight={0: 0, 1: 1}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

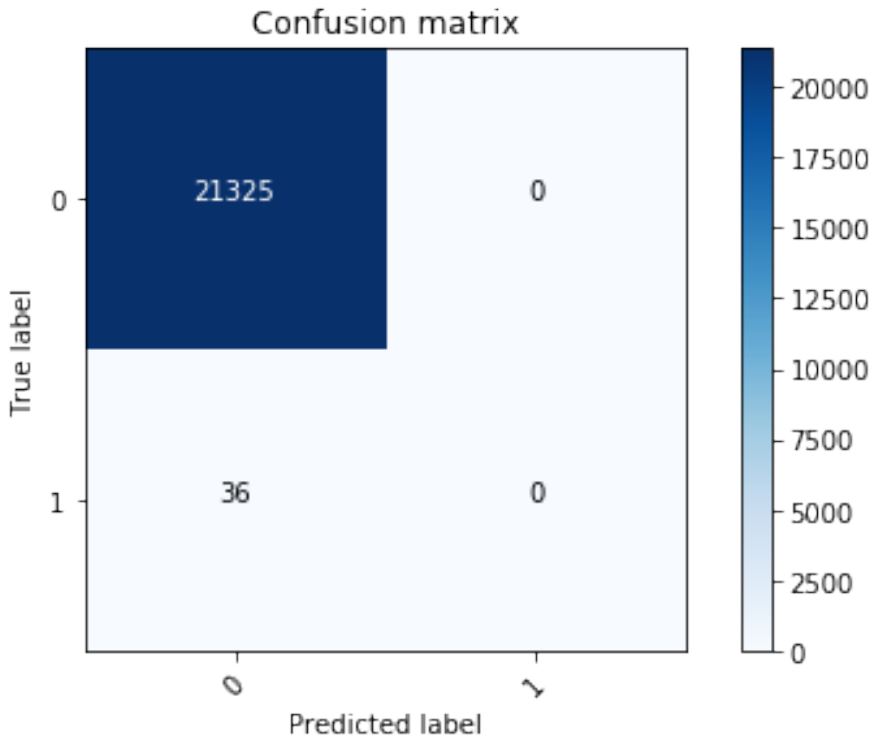
```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[36]: SVC(C=1.0, cache_size=200, class_weight={0: 1, 1: 0}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

In [37]:
```
prediction_SVM_w1 = classifier_w1.predict(x_val) #We predict all the data set.
cm = confusion_matrix(y_val, prediction_SVM_w1)
plot_confusion_matrix(cm,class_names)
```



In [38]:
```
prediction_SVM_w2 = classifier_w2.predict(x_val) #We predict all the data set.
cm = confusion_matrix(y_val, prediction_SVM_w2)
plot_confusion_matrix(cm,class_names)
```




```
In [39]: def frange(start, stop, step):
             i = start
             while i < stop:
                  yield i
                  i += step
         ciertofalso=[]
         falsocierto=[]
         for i in frange(0, 1.0, 0.01):
             j=1-round(i,2)
             classifier = svm.SVC(kernel='linear',class_weight={0:i, 1:j})
             classifier.fit(x_train[0:10000], y_train[0:10000]) # Then we train our model, with
             prediction_SVM_all = classifier.predict(x_val) #We predict all the data set.
             cm = confusion_matrix(y_val, prediction_SVM_all)
             ciertofalso.append(cm[0,1])
             falsocierto.append(cm[1,0])
         ciertofalso
         falsocierto
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0, 1: 1}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
```

```
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.01, 1: 0.99}, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.02, 1: 0.98}, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.03, 1: 0.97}, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.04, 1: 0.96}, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.05, 1: 0.95}, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.060000000000000005, 1: 0.94},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.07, 1: 0.9299999999999999},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.08, 1: 0.92}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.09, 1: 0.91}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.09999999999999999, 1: 0.9},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.10999999999999999, 1: 0.89},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.11999999999999998, 1: 0.88},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.12999999999999998, 1: 0.87},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.13999999999999999, 1: 0.86},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.15, 1: 0.85}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.16, 1: 0.84}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.17, 1: 0.83}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.18000000000000002, 1: 0.8200000000000001}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.19000000000000003, 1: 0.81},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.20000000000000004, 1: 0.8},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.21000000000000005, 1: 0.79},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.22000000000000006, 1: 0.78},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.23000000000000007, 1: 0.77},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.24000000000000007, 1: 0.76},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.25000000000000006, 1: 0.75},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.26000000000000006, 1: 0.74},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.2700000000000001, 1: 0.73},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.2800000000000001, 1: 0.72},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.2900000000000001, 1: 0.71},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.3000000000000001, 1: 0.7},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.3100000000000001, 1: 0.69},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.3200000000000001, 1: 0.6799999999999999}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.3300000000000001, 1: 0.6699999999999999}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.34000000000000014, 1: 0.6599999999999999}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.35000000000000014, 1: 0.65},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.36000000000000015, 1: 0.64},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.37000000000000016, 1: 0.63},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.38000000000000017, 1: 0.62},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.3900000000000002, 1: 0.61},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.4000000000000002, 1: 0.6},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.4100000000000002, 1: 0.5900000000000001}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.4200000000000002, 1: 0.5800000000000001}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.4300000000000002, 1: 0.5700000000000001}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.4400000000000002, 1: 0.56},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.45000000000000023, 1: 0.55},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.46000000000000024, 1: 0.54},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.47000000000000025, 1: 0.53},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.48000000000000026, 1: 0.52},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.49000000000000027, 1: 0.51},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.5000000000000002, 1: 0.5},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.5100000000000002, 1: 0.49},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.5200000000000002, 1: 0.48},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.5300000000000002, 1: 0.47},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.5400000000000003, 1: 0.45999999999999996}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.5500000000000003, 1: 0.44999999999999996}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.5600000000000003, 1: 0.43999999999999995}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.5700000000000003, 1: 0.43000000000000005}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.5800000000000003, 1: 0.42000000000000004}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.5900000000000003, 1: 0.41000000000000003}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.6000000000000003, 1: 0.4},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.6100000000000003, 1: 0.39},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.6200000000000003, 1: 0.38},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.6300000000000003, 1: 0.37},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.6400000000000003, 1: 0.36},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.6500000000000004, 1: 0.35},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.6600000000000004, 1: 0.33999999999999997}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.6700000000000004, 1: 0.32999999999999996}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.6800000000000004, 1: 0.31999999999999995}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.6900000000000004, 1: 0.31000000000000005}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.7000000000000004, 1: 0.30000000000000004}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.7100000000000004, 1: 0.29000000000000004}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.7200000000000004, 1: 0.28},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.7300000000000004, 1: 0.27},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.7400000000000004, 1: 0.26},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.7500000000000004, 1: 0.25},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.7600000000000005, 1: 0.24},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.7700000000000005, 1: 0.22999999999999998}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.7800000000000005, 1: 0.21999999999999997}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.7900000000000005, 1: 0.20999999999999996}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8000000000000005, 1: 0.19999999999999996}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8100000000000005, 1: 0.18999999999999995}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8200000000000005, 1: 0.18000000000000005}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8300000000000005, 1: 0.17000000000000004}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8400000000000005, 1: 0.16000000000000003}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8500000000000005, 1: 0.15000000000000002}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.8600000000000005, 1: 0.14},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.8700000000000006, 1: 0.13},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200, class_weight={0: 0.8800000000000006, 1: 0.12},
           coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.8900000000000006, 1: 0.10999999999999999}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9000000000000006, 1: 0.09999999999999998}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9100000000000006, 1: 0.08999999999999997}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9200000000000006, 1: 0.07999999999999996}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9300000000000006, 1: 0.06999999999999995}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9400000000000006, 1: 0.06000000000000005}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9500000000000006, 1: 0.05000000000000044}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9600000000000006, 1: 0.04000000000000036}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9700000000000006, 1: 0.030000000000000027}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9800000000000006, 1: 0.020000000000000018}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)


Out[39]: SVC(C=1.0, cache_size=200,
           class_weight={0: 0.9900000000000007, 1: 0.010000000000000009}, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

Out[39]: [21325,
          62,
          26,
          23,
          20,
          13,
          10,
          9,
          9,
          9,
          9,
          9,
          9,
          9,
          9,
          10,
          9,
          9,
          10,
          14,
          14,
          14,
          14,
```

```
14,
14,
14,
13,
13,
13,
13,
13,
13,
13,
11,
11,
11,
11,
11,
11,
10,
10,
10,
10,
10,
10,
10,
10,
10,
10,
10,
9,
9,
9,
10,
10,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
7,
7,
7,
5,
5,
6,
6,
6,
```

```
            6,
            5,
            5,
            5,
            4,
            3,
            4,
            4,
            5,
            5,
            4,
            4,
            4,
            4,
            4,
            4,
            4,
            4,
            3,
            3,
            3,
            3,
            3,
            3,
            3,
            3,
            2,
            2,
            2,
            2]

Out[39]: [0,
            3,
            3,
            3,
            4,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            5,
            9,
```

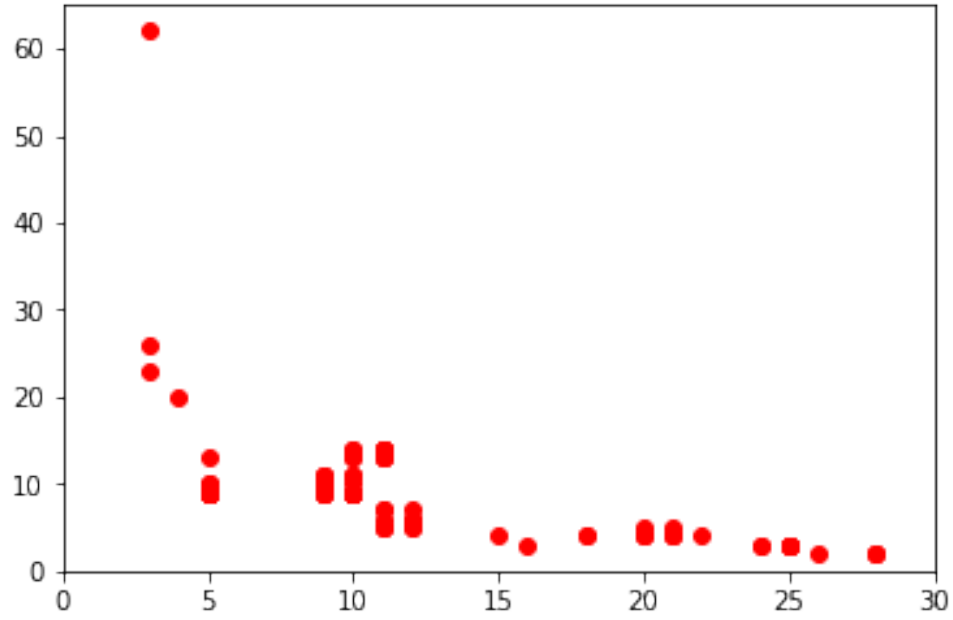
```
10,
10,
10,
11,
11,
11,
11,
11,
11,
11,
11,
11,
11,
10,
10,
10,
10,
10,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
9,
10,
10,
10,
10,
10,
10,
10,
10,
10,
10,
11,
11,
12,
```

```
          12,
          11,
          11,
          12,
          12,
          12,
          11,
          11,
          12,
          15,
          16,
          18,
          18,
          21,
          20,
          21,
          20,
          20,
          20,
          21,
          21,
          22,
          24,
          24,
          25,
          25,
          25,
          25,
          25,
          25,
          26,
          28,
          28,
          28]
```

```
In [40]: plt.plot(falsocierto, ciertofalso, 'ro')
         plt.axis([0, 30, 0, 65])
         plt.show()
```

```
Out[40]: [<matplotlib.lines.Line2D at 0x22e222870f0>]
```

```
Out[40]: [0, 30, 0, 65]
```



- **GridSearch: Parámetros C y gamma óptimos**

```
In [41]: from sklearn import svm, grid_search
         from sklearn.grid_search import GridSearchCV
         def svc_param_selection(X, y, nfolds):
             Cs = [0.001, 0.01, 0.1, 1, 10]
             gammas = [0.001, 0.01, 0.1, 1]
             param_grid = {'C': Cs, 'gamma' : gammas}
             grid_search = GridSearchCV(classifierR , param_grid, cv=nfolds)
             grid_search.fit(X, y)
             grid_search.best_params_
             return grid_search.best_params_
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: T
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: This m
  DeprecationWarning)
```

```
In [42]: svc_param_selection(x_train[0:1000], y_train['Class'][0:1000], 10)
```

```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\cross_validation.py:553: Warning: The least p
  % (min_labels, self.n_folds)), Warning)
```

```
Out[42]: {'C': 0.001, 'gamma': 0.001}
```

```
In [43]: svc_param_selection(x_train[0:10000], y_train['Class'][0:10000], 10)
```

```
Out[43]: {'C': 1, 'gamma': 0.001}
```

```
In [44]: classifier_opt = svm.SVC(C=1, kernel='rbf', gamma=0.001)
```

```
In [47]: tic = time.process_time()
         classifier_opt.fit(x_train, y_train)
         toc = time.process_time()

         time_linear = toc - tic
         print(time_linear)
```

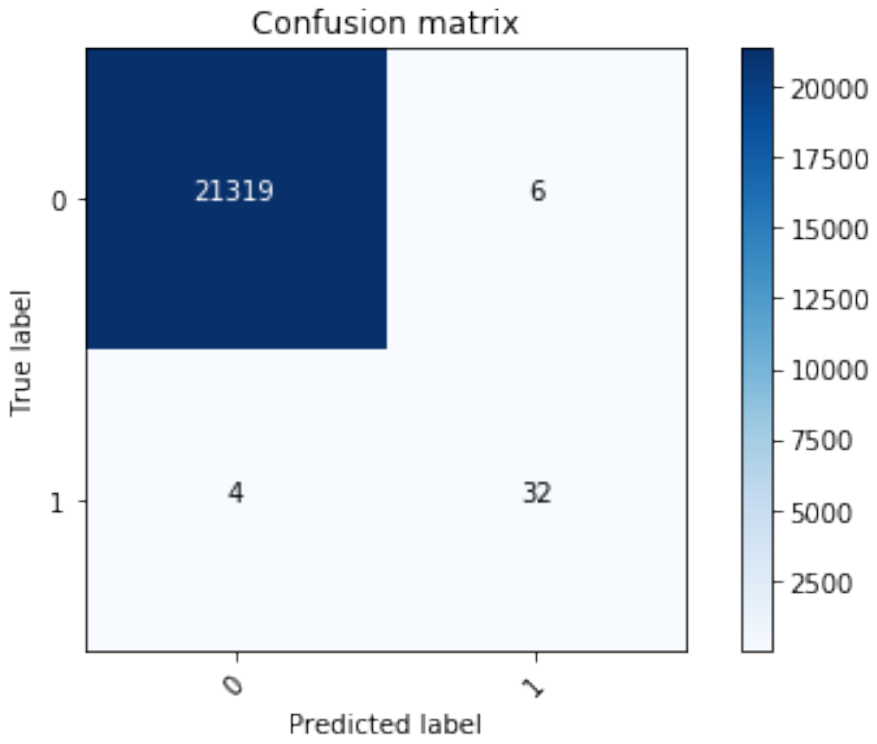
```
C:\Users\ruben\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarnin
  y = column_or_1d(y, warn=True)
```

```
Out[47]: SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

```
448.890625
```

```
In [48]: prediction_SVM_opt = classifier_opt.predict(x_val)
```

```
In [49]: cm_opt = confusion_matrix(y_val, prediction_SVM_opt)
         plot_confusion_matrix(cm_opt, class_names)
```



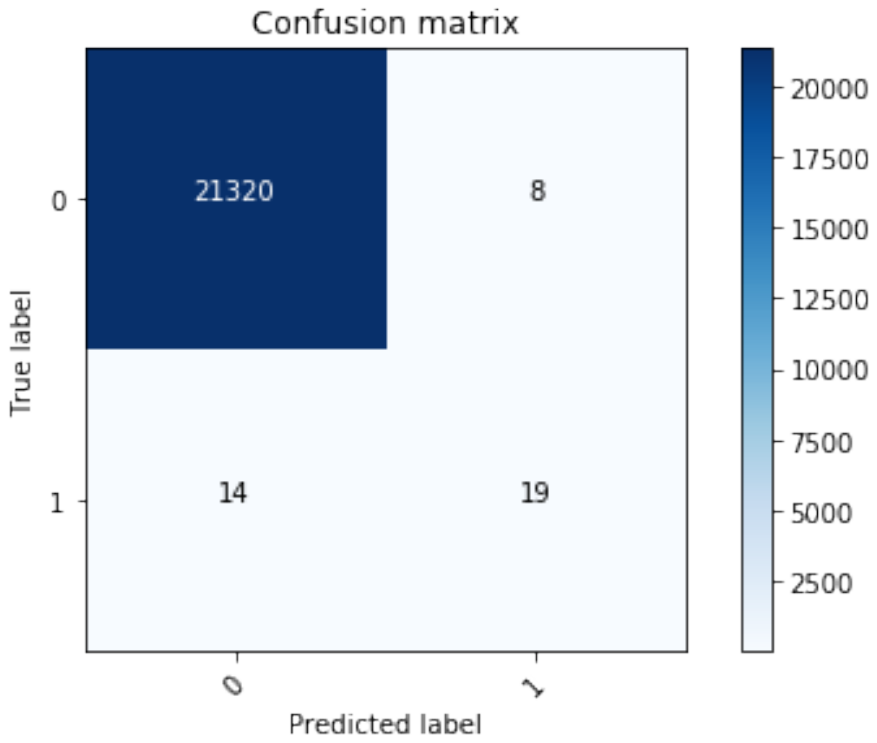
```
In [50]: #Performance measurement
         opt_model_accuracy = accuracy_score(y_val, prediction_SVM_opt)
         opt_model_recall = recall_score(y_val, prediction_SVM_opt)
         opt_model_precision = precision_score(y_val, prediction_SVM_opt)
         opt_model_f1score = f1_score(y_val, prediction_SVM_opt)
         opt_model_roc_auc_score = roc_auc_score(y_val, prediction_SVM_opt)

         print("The Accuracy is", opt_model_accuracy) #acertar la prediccion en general
         print("The Precision is",opt_model_precision )
         print("The Recall is", opt_model_recall)
         print("The F1 Score is",opt_model_f1score)
         print("ROCAUC:", opt_model_roc_auc_score)
```

```
The Accuracy is 0.9995318571227939
The Precision is 0.8421052631578947
The Recall is 0.8888888888888888
The F1 Score is 0.8648648648648649
ROCAUC: 0.9443037644913378
```

0.6 Apply model to Test Set: RBF Kernel, C=1, gamma=0.001

```
In [51]: prediction_SVM_test = classifier_opt.predict(x_test)
```

```
In [52]: cm_test = confusion_matrix(y_test, prediction_SVM_test)
         plot_confusion_matrix(cm_test, class_names)
```



In [53]:
```
#Performance measurement
svm_model_accuracy = accuracy_score(y_test, prediction_SVM_test)
svm_model_recall = recall_score(y_test, prediction_SVM_test)
svm_model_precision = precision_score(y_test, prediction_SVM_test)
svm_model_f1score = f1_score(y_test, prediction_SVM_test)
svm_model_roc_auc_score = roc_auc_score(y_test, prediction_SVM_test)

print("The Accuracy is", svm_model_accuracy) #acertar la prediccion en general
print("The Precision is", svm_model_precision)
print("The Recall is",svm_model_recall)
print("The F1 Score is",svm_model_f1score)
print("ROCAUC:", svm_model_roc_auc_score)
```

```
The Accuracy is 0.9989700856701466
The Precision is 0.7037037037037037
The Recall is 0.5757575757575758
The F1 Score is 0.6333333333333334
ROCAUC: 0.7876912409920662
```

Anexo E

Mínimos de una función

A la hora de resolver un problema de optimización, donde se pretende minimizar la función objetivo, es esencial conocer la naturaleza de dicha función para poder hallar sus mínimos.

Para que el *Gradient Descent* sea eficiente y encuentre el mínimo global, es necesario que la función *cost* incluida en $J(\theta)$ sea convexa. En matemática, una función real es convexa en un intervalo (a,c), si para todo punto b del intervalo la recta tangente en ese punto queda por debajo de la función. Una de las propiedades más interesantes, de las funciones estrictamente convexas, es que sólo tienen un mínimo local y que se trata de su mínimo absoluto. Por lo tanto, si nuestra función de costes es convexa, hallaremos su mínimo absoluto.

En el caso que la función de costes no fuera convexa, sucedería el escenario de la figura E.1. Se aprecia que existen varios mínimos y que según los parámetros θ escogidos inicialmente, se llegará a un mínimo u otro.

Por otra parte, si nuestra función de costes es convexa, hallaremos el mínimo global ya que no dependerá de los parámetros θ escogidos inicialmente (e.g. Figura E.2, paraboloide elíptico).

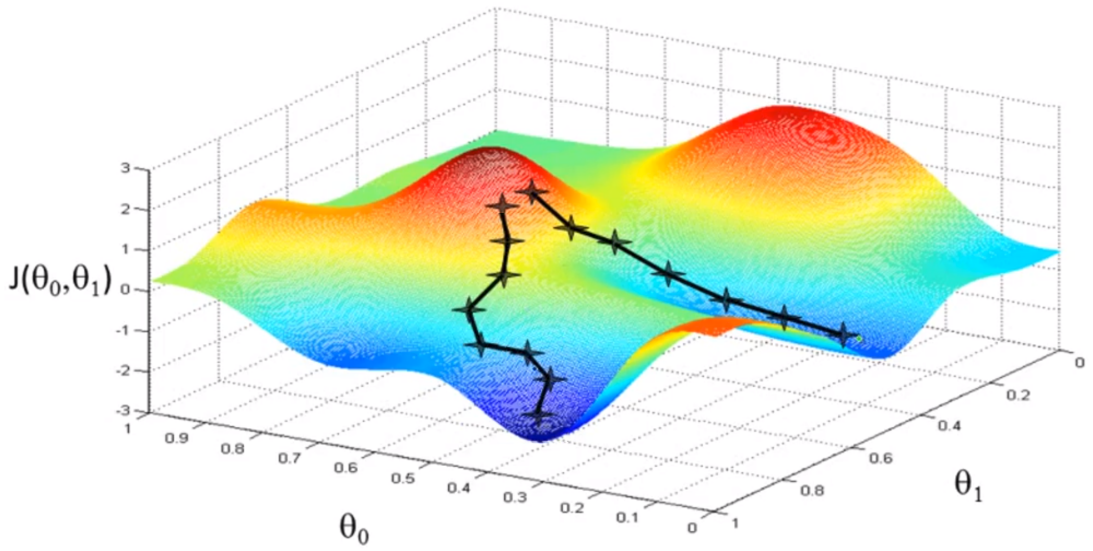


FIGURA E.1: Función de costes no convexa

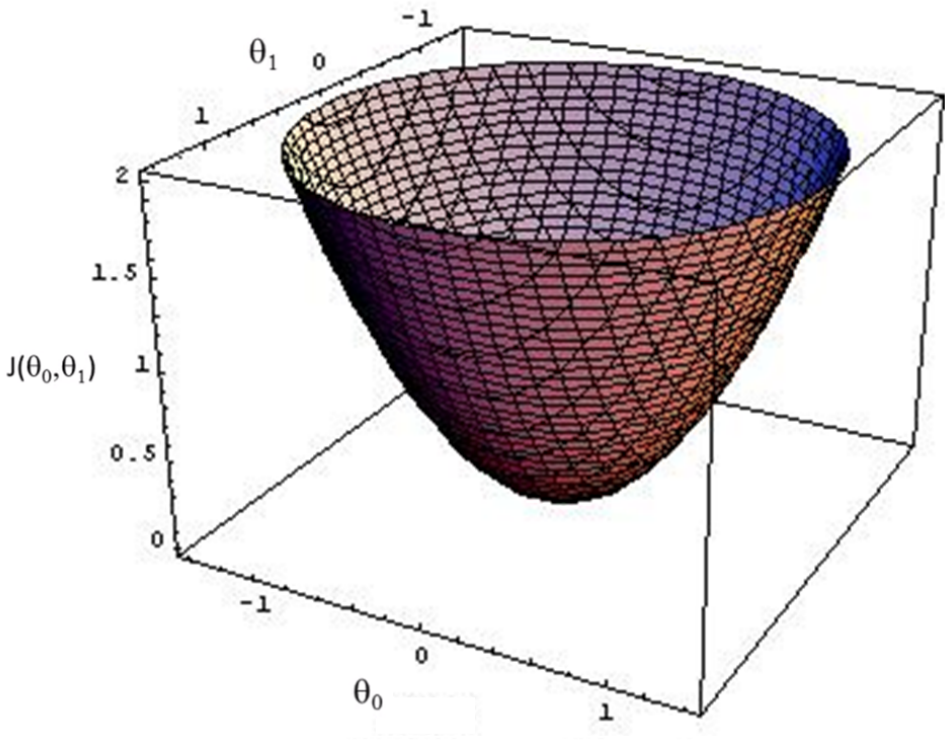


FIGURA E.2: Función de costes convexa

Anexo F

Derivada parcial de la función de costes en regresión logística

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \left(\log(h_\theta \left(x^{(i)} \right) \right) + (1 - y^{(i)}) \left(\log(1 - h_\theta \left(x^{(i)} \right) \right) \right]$$

$$\underset{\text{linearidad}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\partial}{\partial \theta_j} \log \left(h_\theta \left(x^{(i)} \right) \right) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} \left(\log(1 - h_\theta \left(x^{(i)} \right) \right) \right]$$

$$\underset{\text{regla de la cadena}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} (h_\theta \left(x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} \left(1 - h_\theta \left(x^{(i)} \right) \right)}{1 - h_\theta \left(x^{(i)} \right)} \right]$$

$$\underset{h_\theta(x) = \sigma\left(\theta^\top x\right)}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} \sigma\left(\theta^\top x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} + (1 - y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} \left(1 - \sigma\left(\theta^\top x^{(i)} \right) \right)}{1 - h_\theta \left(x^{(i)} \right)} \right]$$

$$\underset{\sigma'}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\sigma\left(\theta^\top x^{(i)} \right) \left(1 - \sigma\left(\theta^\top x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} - (1 - y^{(i)}) \frac{\sigma\left(\theta^\top x^{(i)} \right) \left(1 - \sigma\left(\theta^\top x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{1 - h_\theta \left(x^{(i)} \right)} \right]$$

$$\underset{\sigma\left(\theta^\top x\right) = h_\theta(x)}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{h_\theta \left(x^{(i)} \right) \left(1 - h_\theta \left(x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{h_\theta \left(x^{(i)} \right)} - (1 - y^{(i)}) \frac{h_\theta \left(x^{(i)} \right) \left(1 - h_\theta \left(x^{(i)} \right) \right) \frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right)}{1 - h_\theta \left(x^{(i)} \right)} \right]$$

$$\underset{\frac{\partial}{\partial \theta_j} \left(\theta^\top x^{(i)} \right) = x_j^{(i)}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \left(1 - h_\theta \left(x^{(i)} \right) \right) x_j^{(i)} - \left(1 - y^i \right) h_\theta \left(x^{(i)} \right) x_j^{(i)} \right]$$

$$\underset{\text{distributividad}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^i - y^i h_\theta \left(x^{(i)} \right) - h_\theta \left(x^{(i)} \right) + y^{(i)} h_\theta \left(x^{(i)} \right) \right] x_j^{(i)}$$

$$\underset{\text{cancelación}}{=} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} - h_\theta \left(x^{(i)} \right) \right] x_j^{(i)}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta \left(x^{(i)} \right) - y^{(i)} \right] x_j^{(i)}$$

Anexo G

Parámetro de aprendizaje α

El parámetro de aprendizaje α forma parte del algoritmo de *Gradient Descent* y se encarga de ajustar el módulo de los pasos que dan los parámetros θ a cada iteración.

Repeat {
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
}

¿Que sucede si el parámetro de aprendizaje no está bien ajustado?
Existen dos escenarios no deseados, en primer lugar, que la α sea demasiado pequeña. En esta situación, los pasos que dan los parámetros θ serán muy pequeños. Retomando el ejemplo de la figura 4.13, sucedería lo que se puede apreciar en la figura G.1.

En este caso, el algoritmo puede ser muy lento, puesto que debe realizar una gran cantidad de pasos para llegar al óptimo.

Por otra parte, si la α es demasiado grande, el problema está en que el algoritmo no logre converger (se salte el óptimo) o incluso diverja (Figura G.2).

Por lo tanto, es de suma importancia escoger un parámetro de aprendizaje adecuado para la ejecución del *Gradient Descent*.

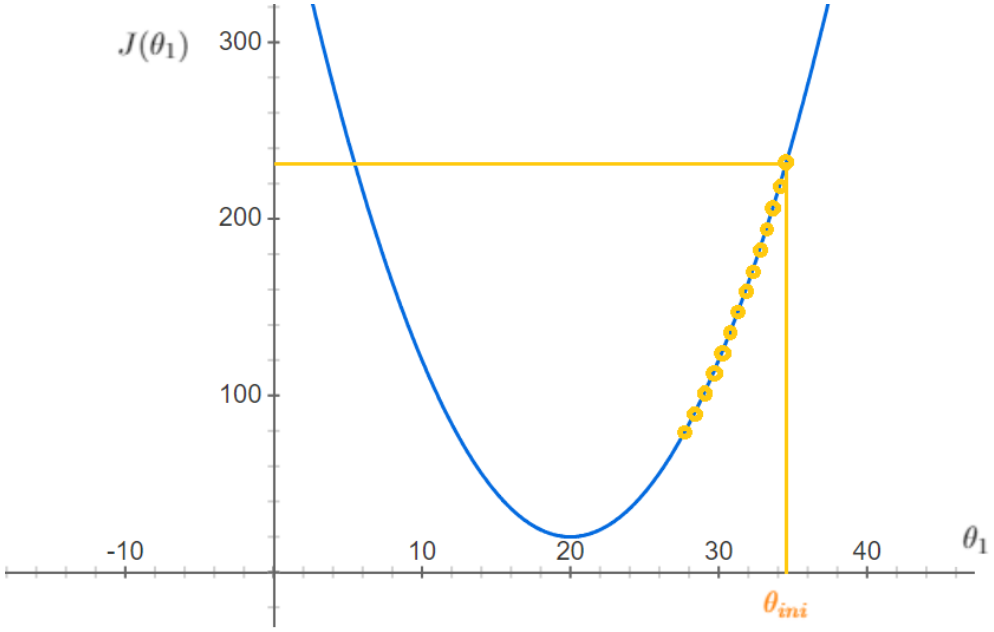


FIGURA G.1: Parámetro de aprendizaje demasiado pequeño

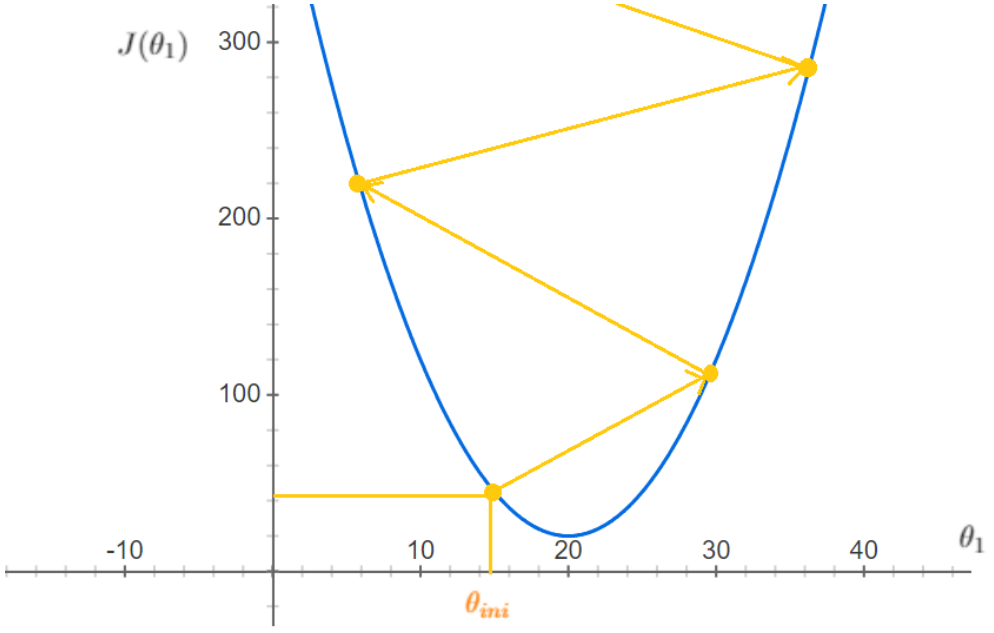


FIGURA G.2: Parámetro de aprendizaje demasiado grande

Anexo H

Código exploración dataset

En el siguiente anexo se detalla el código empleado para explorar el conjunto de datos de estudio. El análisis de dicha exploración se detalla en el capítulo 4.

In [1]:
```
import numpy as np # biblioteca de funciones matemáticas de alto nivel para operar
                   #con esos vectores o matrices.
import pandas as pd # ofrece estructuras de datos y operaciones para manipular tablas nu
import seaborn as sns # for intractve graphs
import sklearn #bibliothèque dédiée à l'apprentissage automatique. Fonctions pour estime
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_
import itertools

%matplotlib inline
#guardar los graficos

import matplotlib #biblioteca para la generación de gráficos a partir de datos contenido
import matplotlib.pyplot as plt #plt.plot(x, y) implica coord y ord grafico
```

In [18]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Matriz de confusión',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.show()

def show_data(cm, print_res = 0):
    tp = cm[1,1]
    fn = cm[1,0]
    fp = cm[0,1]
    tn = cm[0,0]
    if print_res == 1:
```

```
            print(tp)
            print(tn)
            print(fp)
            print(fn)
            print('Accuracy =      {:.3f}'.format((tn+tp)/(tp+fp+tn+fn)))
            print('Precision =     {:.3f}'.format(tp/(tp+fp)))
            print('Recall (TPR) =  {:.3f}'.format(tp/(tp+fn)))
            print('Fallout (FPR) = {:.3e}'.format(fp/(fp+tn)))
        return tp/(tp+fp), tp/(tp+fn), fp/(fp+tn)
```

In [3]:
```
data = pd.read_csv("creditcard.csv") #subir los datos
len(data)
```

Out[3]:
```
284807
```

In [4]:
```
data.isnull().sum().max()
```

Out[4]:
```
0
```

In [5]:
```
count_class=data.Class.value_counts()
count_class
```

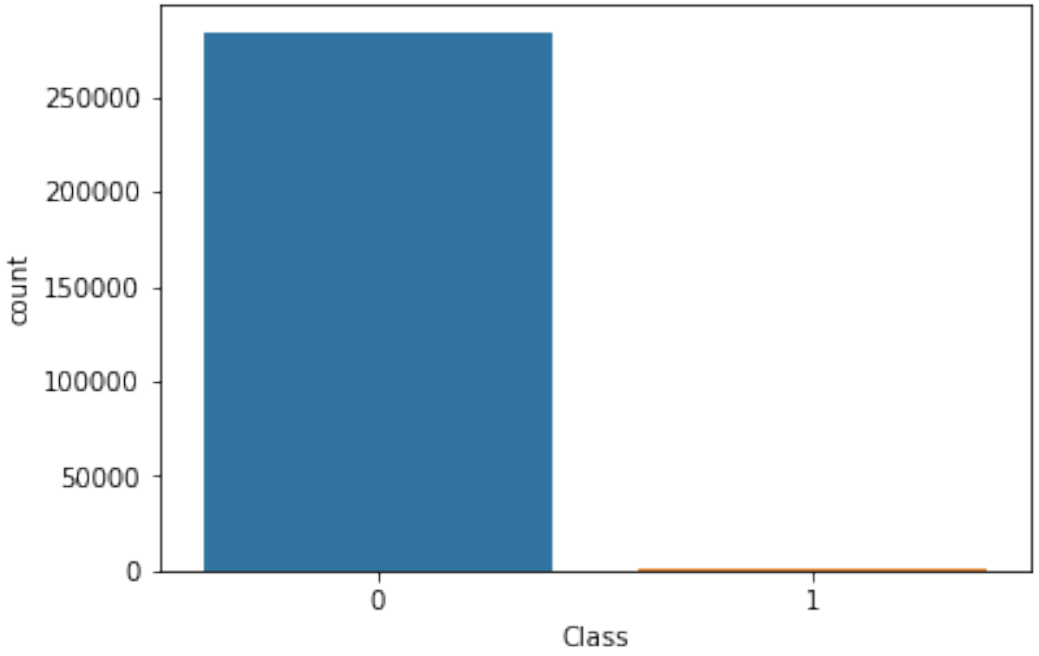
Out[5]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [6]:
```
data.columns
#data.head()
#data.tail()
```

Out[6]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [7]:
```
# Now lets check the class distributions
sns.countplot("Class",data=data)
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x234078aa198>
```




```
In [8]: data_fraud = data[data['Class'] == 1] # Recovery of fraud data
        data_nfraud = data[data['Class'] == 0]

        # Count and %
        Count_Normal_transacation = len(data[data['Class']==0])
        Count_Fraud_transacation = len(data[data['Class']==1])
        Percentage_of_Normal_transacation = Count_Normal_transacation/(Count_Normal_transacation
        print('% of normal transacation       :', Percentage_of_Normal_transacation*100)
        print('Number of normal transaction   :', Count_Normal_transacation)
        Percentage_of_Fraud_transacation= Count_Fraud_transacation/(Count_Normal_transacation+Co
        print('% of fraud transacation        :',Percentage_of_Fraud_transacation*100)
        print('Number of fraud transaction    :', Count_Fraud_transacation)
```

```
% of normal transacation       : 99.82725143693798
Number of normal transaction   : 284315
% of fraud transacation        : 0.1727485630620034
Number of fraud transaction    : 492
```

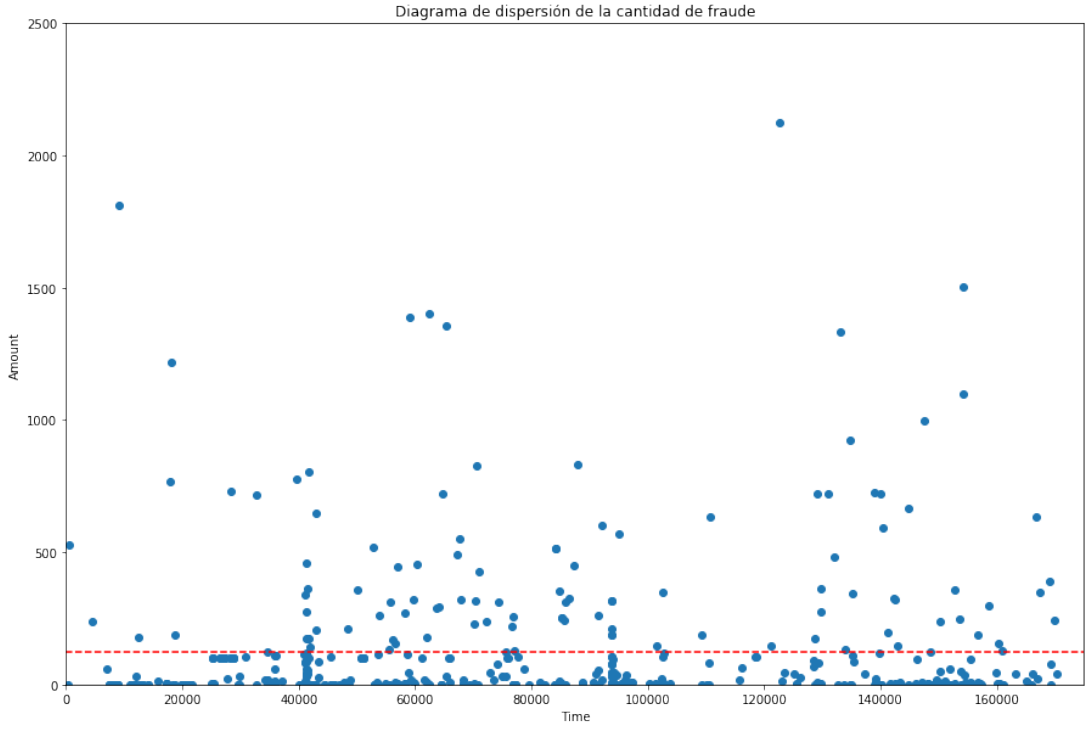
```
In [9]: data.describe() #te da la suma, media, min max etc
```

```
Out[9]:                 Time            V1            V2            V3            V4  \
        count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
        mean    94813.859575  3.919560e-15  5.688174e-16 -8.769071e-15  2.782312e-15
        std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
```

```
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552563e-15  2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

           ...                V21           V22           V23           V24  \
count      ...       2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean       ...       1.537294e-16  7.959909e-16  5.367590e-16  4.458112e-15
std        ...       7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min        ...      -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%        ...      -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%        ...      -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%        ...       1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max        ...       2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

In [10]:
```
plt.figure(figsize=(15,10))
plt.scatter(data_fraud['Time'], data_fraud['Amount']) # Display fraud amounts according
plt.title('Diagrama de dispersión de la cantidad de fraude')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.xlim([0,175000])
plt.ylim([0,2500])
pinny_mean= data_fraud['Amount'].mean()
plt.axhline(y=pinny_mean, label='Amount Mean', linestyle='--', color='red')
pinny_mean
```

Out[10]: 122.21132113821133



In [11]:
```
data_fraud['Amount'].max()
```

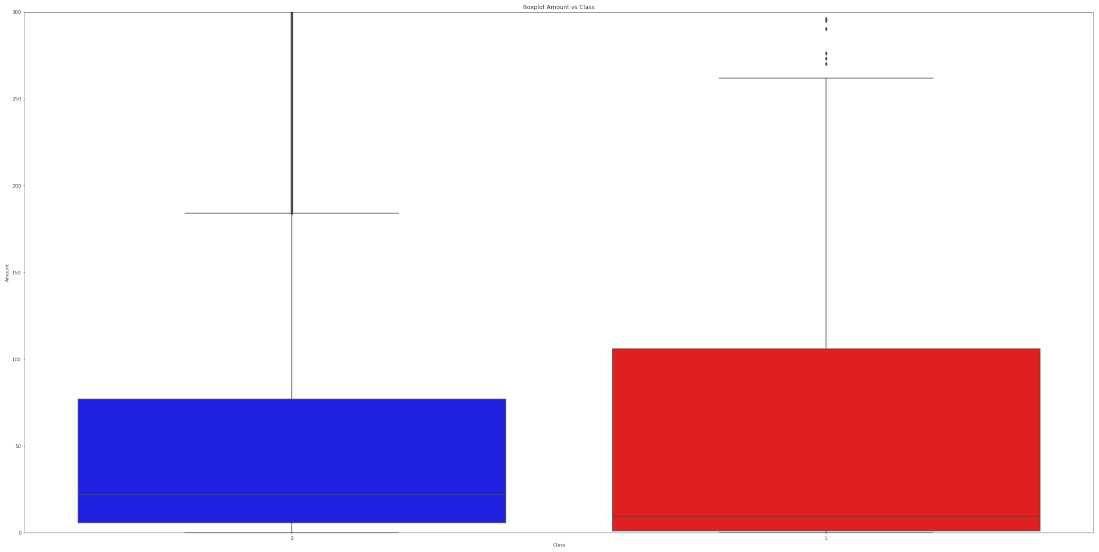
Out[11]: 2125.87

In [12]:
```
data_fraud['Amount'].min()
```

Out[12]: 0.0

In [13]:
```
my_pal = {0: 'blue', 1: 'red'}
plt.figure(figsize = (40, 20))
ax = sns.boxplot(x = 'Class', y = 'Amount', data = data, palette = my_pal)
```

```
ax.set_ylim([0, 300])
plt.title('Boxplot Amount vs Class')
plt.show()
```



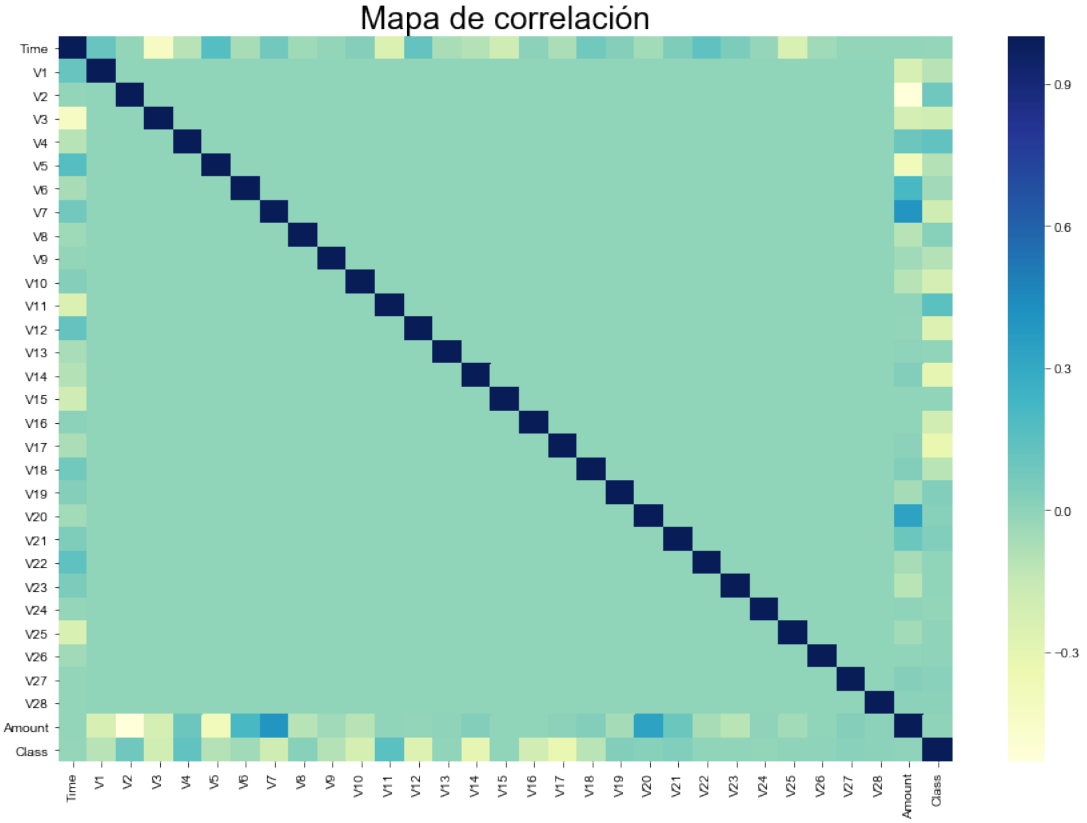

```
In [14]: print(str(sum(data_fraud['Amount']))+'UM han sido defraudados y'+ str(sum(data_nfraud['
```

```
60127.96999999997UM han sido defraudados y25102462.039983638 no han sido defraudados
```

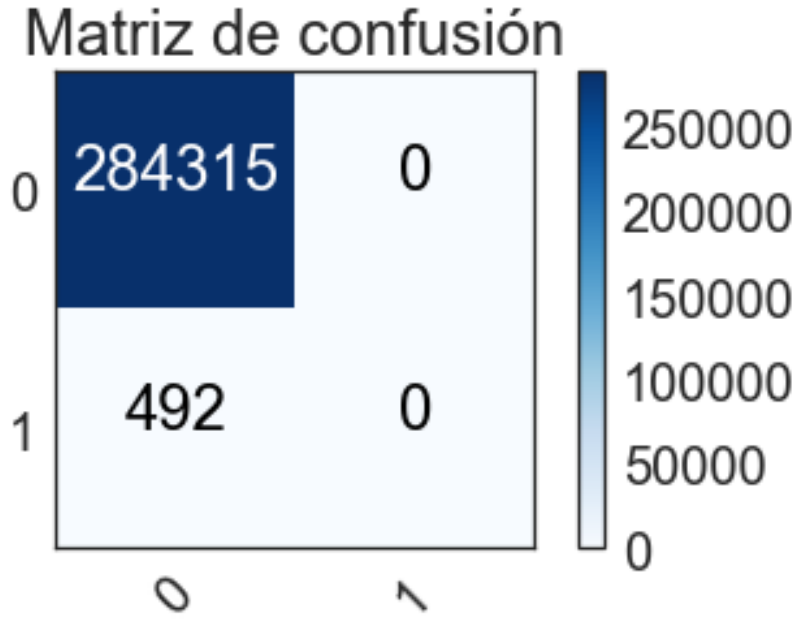
```
In [15]: print("The accuracy of the classifier then would be : "+ str((284315-492)/284315)+ " wh
```

```
The accuracy of the classifier then would be : 0.998269524998681 which is the number of good cla
```

```
In [16]: data_corr = data.corr()
         plt.figure(figsize=(15,10))
         sns.heatmap(data_corr, cmap="YlGnBu") # Displaying the Heatmap
         sns.set(font_scale=2,style='white')

         plt.title('Mapa de correlación')
         plt.show()
```




```
In [19]: y = data.loc[:, data.columns == 'Class']
         y0ss=[]
         for i in range(284807):
             y0ss.append(0)
         y0s = {'col1': y0ss}
         y0 = pd.DataFrame(data=y0s)
         cm = confusion_matrix(y, y0)
         plot_confusion_matrix(cm, ['0', '1'], )
```


Matriz de confusión
284315
0
492
0
0
1
0
1
250000
200000
150000
100000
50000
0

Bibliografía

A. Ng “Machine Learning” in Coursera

J. H. Friedman, “Data mining and statistics: What’s the connection,” in Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics, 1997.

B. P. Battula and R. S. Prasad, “An Overview of Recent Machine Learning Strategies in Data Mining,” International Journal of Advanced Computer Science and Applications, vol. 4, 2013.

C. Sammut and G. I. Webb, eds., Encyclopedia of Machine Learning. Springer, 2010.

S. Salzberg, “Book review: C4.5: programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993,” Machine Learning, vol. 16, no. 3, pp. 235–240, 1993.

Towardsdatascience. “Support Vector Machines” [En línea]. < https://towardsdatascience.com/support-vector-machines-a-brief-overview-37e018ae310f>

Peopleplusprofit “Industria 4.0” [En línea]. < https://peopleplusprofit.org/industria-4-0-apuesta-por-la-sostenibilidad/>